МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Менеджмент разработки программного обеспечения
Конспект лекций

Электронное учебное пособие

САМАРА
2011

УДК 004.9 (075)
ББК 32.9я7
К 142

Автор-составитель: **Казанский Николай Львович**

Менеджмент разработки программного обеспечения. Конспект лекций

[Электронный ресурс] : электрон. учеб. пособие / Н.Л. Казанский; М-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). – Электрон. текстовые и граф. дан. (2,9 Мбайт). - Самара, 2011. - 1 эл. опт. диск (CD-ROM).

Рассмотрены базовые принципы, методики и приемы управления процессами разработки коммерческого программного обеспечения. Проанализирован реальный опыт успешной разработки коммерческого программного обеспечения в небольшой начинающей компании.

Учебное пособие предназначено для подготовки магистров направления 010400.68 «Прикладная математика и информатика» факультета информатики, изучающих дисциплину «Менеджмент разработки программного обеспечения» в 9 семестре.

Электронное учебное пособие разработано на кафедре технической кибернетики.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. Королева
(национальный исследовательский университет)»

Н.Л. Казанский

Менеджмент разработки программного обеспечения. Конспект лекций

Электронное учебное пособие

САМАРА 2011

УДК 004.9 (075)
ББК 32.9я7
К 142

Автор: **Казанский Николай Львович**

Казанский Н.Л. Менеджмент разработки программного обеспечения. Конспект лекций [Электронный ресурс] : электрон. учеб. пособие / Н.Л. Казанский; М-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). – Электрон. текстовые и граф. дан. (2,9 Мбайт). - Самара, 2011. -1 эл. опт. диск (CD-ROM).

Режим доступа: http://virtual6.ssau.ru/Moodle/course/view.php?id=35

Рассмотрены базовые принципы, методики и приемы управления процессами разработки коммерческого программного обеспечения. Проанализирован реальный опыт успешной разработки коммерческого программного обеспечения в небольшой начинающей компании.

Учебное пособие предназначено для подготовки магистров направления 010400.68 «Прикладная математика и информатика» факультета информатики, изучающих дисциплину «Менеджмент разработки программного обеспечения» в 9 семестре.

Электронное учебное пособие разработано на кафедре технической кибернетики.

Бизнес-подходы к распространению программного обеспечения

Классификация типов распространения программного обеспечения

В настоящее время общепризнанными являются несколько подходов к распространению программного обеспечения (ПО), определяющие следующую классификацию:

• проприетарное (т.е. полностью коммерческое) ПО;

• «шароварное» ПО (от английского слова “shareware”) – условно-коммерческое;

• свободно-распространяемое ПО;

• некоммерческое;

• ПО с открытым источником.

Первый вариант распространения ПО предполагает чисто коммерческие отношения с жестким пресечением нелицензионного использования ПО, вплоть до уголовного преследования.

Второй вариант («шароварное» ПО) предполагает свободное скачивание программного продукта и его использование потребителя с предложением перевести разработчику оговоренную стоимость продукта после истечения срока тестового использования.

Следующие три варианта распространения ПО предполагают его свободное бесплатное применение (и даже модификацию) в случае соблюдения условий лицензии, оговаривающей правила пользования. При этом лицензия оговаривает достаточно широкий спектр условий распространения.

Например, условия распространения программного обеспечения с открытыми исходниками должны удовлетворять следующим критериям:

А Свободное распространение. Лицензия не должна ограничивать право любой стороны продавать или раздавать программное обеспечение как часть совокупного комплекта программного обеспечения, происходящего из разных источников. Лицензия не должна требовать отчислений или иной платы при таком распространении.

Б. Исходный код. Программа должна включать исходный код и должна допускать распространение в исходных кодах так же, как и в скомпилированной форме. Если в какой-либо форме продукт распространяется без исходного кода, должны быть общедоступны каналы получения исходного кода по цене, не превышающей стоимость копирования, — предпочтительно посредством бесплатной загрузки по Internet. Исходный код должен быть основной формой, в которой программист модифицирует программу. Намеренное запутывание исходного кода не допускается. Замена исходного кода промежуточными формами

(такими, как результат работы препроцессора или транслятора) не допускается.

В. Производные произведения. Лицензия должна допускать модификацию [и создание] производных произведений, а также их распространение на тех же условиях, что и оригинальное ПО.

Г. Целостность авторского исходного кода. Лицензия может ограничивать распространение исходного кода в модифицированной форме, только если она допускает распространение с исходным кодом «заплаток» («patch files») для его модификации во время построения исполняемого кода. Такая лицензия должна явно разрешать распространение исполняемого кода, построенного из модифицированного исходного кода. Такая лицензия может требовать, чтобы производное произведение носило отличающееся от оригинального имя или номер версии.

Д. Отсутствие дискриминации лиц и групп. Лицензия не должна дискриминировать любое лицо или группу лиц.

Е. Отсутствие дискриминации областей использования. Лицензия не должна дискриминировать использование программы в конкретных областях. Например, она не может запрещать коммерческое использование программы или запрещать использование программы при проведении генетических исследований.

Ж. Распространение лицензии. Права, передаваемые с программой, должны передаваться каждому, получившему программу, без необходимости отдельного лицензирования.

З. Лицензия не должна быть специфичной для продукта. Права, передаваемые с программой, не должны зависеть от того, включена ли программа в состав определенной дистрибуции. Если программа выделяется из дистрибуции и используется или распространяется в соответствии с лицензией, все стороны, получившие программу, должны обладать теми же правами, которые переданы пользователям оригинальной дистрибуции.

И. Лицензия не должна ограничивать распространение другого программного обеспечения. Лицензия не должна устанавливать ограничений на другое ПО, распространяемое вместе с лицензируемым ПО. Например, лицензия не должна требовать, чтобы все прочие программы, распространяемые на том же носителе, были программами с открытыми исходниками.

Источник: Open Source Initiative, www.osi.org**.**

Тем не менее, существует достаточно много способов зарабатывания при создании бесплатного ПО. Перечислим их.

Компоненты бизнес-моделей для косвенных продаж :

А. Использование софта с открытым источником для создания рыночной позиции для закрытого софта (пример: Netscape Communications, Inc.).

Б. Использование софта с открытым источником для создания рыночной позиции для оборудования (пример: Apple с их Darwin).

В. Использование софта с открытым источником для создания рыночной позиции для услуг (пример: Zope).

Г. Продажа «официальных аксессуаров» (от маек и кружек до документации) (пример: O'Reilly Associates).

Д. Отсроченное открытие исходников (пример: Aladdin Enterprises с их Ghostscript).

Е. Использование софта с открытым источником для создания / поддержания брэнда.

Ж. Использование софта с открытым источником для торговли оперативно обновляемым контентом.

Бизнес-модели развития компаний-разработчков ПО

В настоящее время общепризнанными являются две модели развития компаний: «израильско-скандинавская» или продуктовая модель, когда компания инвестирует самостоятельно или с помощью инвестиционных фондов в разработку продукта и выведение его на рынок и свои прибыли получает от продажи тиражируемого продукта; и «индийская» или проектная модель, когда компания выполняет заказной проект по разработке программного обеспечения и зарабатывает свою прибыль на каждом таком выполненном проекте.

Несколько слов о каждой модели.

Деньги

Если оценивать по-крупному, то, как в проектной, так и в продуктовой модели бюджеты компаний состоят из бюджетов на разработку программного обеспечения (продукта или проекта) и бюджетов на организацию продаж и маркетинга. Основная разница – в их соотношении. В проектной компании бюджет на разработку, по крайней мере, сравним с бюджетом на продажи и маркетинг. Это связано со следующими факторами:

1. Потребность рынка в компаниях, построенных в соответствии с проектной моделью, в основном определяется хронической недостачей квалифицированных специалистов для исполнения проектов. Потребность определяется уровнем развития информационных технологий в конкретной стране и системой образования, которая готовит специалистов для отрасли. Дисбаланс вызывает потребность. Заметьте, что потребность в основном определяется недостачей специалистов, а не ценой услуг. Это означает, что, специализируясь не только на рынке проектов, но и на его определенной части, где недостача чувствуется особо остро, цена услуг просто не будет иметь значения. Дисбаланс существовал, существует и только увеличивается с развитием информационных технологий. Это означает, что потребность в качественном исполнении проектов будет существовать в обозримом будущем. Могут меняться технологии, инструменты, но потребность будет только усиливаться.

2. Следовательно, деньги из бюджета продаж и маркетинга тратятся в основном на поиск новых заказчиков, у которых по определению существует такая потребность в услугах компании, а не на развитие или тем паче создание рынка и формирование потребности в услугах компании. В продуктовой компании бюджет продаж и маркетинга сильно зависит от того, как продукт «попал в рынок». То есть, если продукт занимает нишу, где нет конкурентов и сам продукт достаточно качественный, он продает сам себя. В этом случае на этапе начальных продаж затраты на их организацию также сравнимы или ниже затрат на разработку. Но таким образом везет очень немногим. В основном продуктовым компаниям приходится делать программные средства там, где уже существуют конкурентные продукты. Следовательно, для того чтобы войти на рынок и отвоевать определенный процент рынка, требуются массированные финансовые вливания в маркетинг и продажи. В этом случае бюджет разработки значительно ниже бюджета отдела продаж и маркетинга.

Разобравшись с бюджетами, поговорим о прибыли. Проект уже делается для заказчика, то есть если нет заказчика, который готов платить деньги, то нет и проекта. Следовательно, конечный элемент проектной компании уже приносит компании прибыль. Следовательно, проектная компания по определению прибыльна, при недопущении менеджментом глупых решений, как то: допущение процента оплачиваемых проектов в компании ниже определенного уровня или значительное увеличение бюджетов не связанных с разработкой. Продукт делается в основном для будущего заказчика, то есть в момент разработки продукта компания является неприбыльной, далее вхождение на рынок и организации продаж также требует значительных вложений. То есть существует замкнутый круг. Продуктовая компания старается быть прибыльной, а для того чтобы достичь прибыльности, она должна тратить больше денег.

Размер прибыли проектной компании относительно предопределен, так как на выполнение каждого проекта есть определенные заданные расходы и обороты наращиваются экстенсивным путем, то есть увеличение количества сотрудников для исполнения проектов. Изменять прибыль компании можно только увеличением загрузки компании и увеличением цены, но нужно отметить, что серьезно увеличить ее нельзя. Размер прибыли продуктовой компании не ограничен сверху ничем кроме количества заказчиков, которые могут купить продукт. При правильном попадании в рынок можно обеспечить сотни, а иногда и тысячи процентов прибыли.

Отсюда основной вывод:

1. Проектная компания при правильном управлении является прибыльной и может существовать и развиваться на собственные средства.

2. Продуктовая компания по определению является инвестиционной и может развиваться на начальном этапе, на инвестиционные деньги.

Риски

Исходя из того, что продуктовая компания серьезно зависит от неопределенности рынка, она является более рисковой, чем проектная

компания. Даже обязательное применение методик исследования рынков не может гарантировать снижение основного риска до уровня проектной компании, где рынок более или менее определен.

Если говорить о рисках другого порядка, можно отметить, что проектная компания обладает большими рисками при сравнении с продуктовой компанией такого же оборота в следующих областях:

- риски управления компанией. Проектная компания обладает большим персоналом для достижения такого же оборота. Следовательно, управление проектной компанией более сложное и несет в себе большие риски;
- риски роста. Так как проектная компания растет экстенсивным путем, то есть путем увеличения персонала, то рост компании, особенно если требуется значительные скорости роста, сопряжен с рисками. Потому что значительное увеличение численности компании без построения соответствующей системы управления и особенно управления качеством, может привести к потере качества исполнения проектов и потере заказчиков, а, следовательно, и прибыли;
- риски, связанные с заказчиками. При неправильном управлении количеством и размерностью заказчиков, в проектной компании, может образоваться перекос доходов в сторону одного заказчика, что может привести к серьезным проблемам компании, если этот заказчик прекратит работать с компанией;
- страновые риски. Так как проектная компания обладает большой размерностью и меньшей мобильностью производственной составляющей и, кроме того, производство в проектных компаниях обычно располагается в странах с меньшим уровнем жизни и большими политическими и экономическими рисками, следовательно, она обладает большими страновыми рисками.

Что в результате?

Необходимо отметить, что продуктовая модель при правильной реализации, правильных инвесторах и попадании в рынок может достичь гораздо больших результатов, чем проектные компании. Это выражается в:

1. большей прибыли компании;
2. большей оценке компании при инвестициях, так как оценивается в основном потенциал компании, а не финансовые результаты;
3. большей рыночной капитализации компании.

Выводы

1. Продуктовая модель может действительно приносить большие финансовые результаты при меньших управленческих затратах.
2. Продуктовая модель в подавляющем числе случаев требует значительно больших инвестиций для развития.
3. Продуктовая модель не может применяться в массовом порядке без присутствия достаточного количества инвесторов.
4. Продуктовая модель имеет большие риски в своей природе, поэтому инвестор должен быть не стандартным, а венчурным.

Что надо и что есть сейчас в России?

Для массового развития продуктовой модели в России необходимы следующие условия:

1. Наличие менеджеров, которые понимают западный рынок и то, что ему нужно. Несмотря на отдельные успехи компаний, ориентированных на отечественный рынок («1С», «Яндекс» и т.п.) основные средства можно получить только после выхода на глобальные рынки. Если мы говорим об экспортном потенциале, то продукты должны продаваться вне России, если конкретизировать, то на рынках Северной Америки, Европы и Японии – это основные платежеспособные рынки. Очень трудно не понимая потребителей придумывать идеи, которые им могут быть проданы после воплощения в продукт. Кроме того, необходимо, чтобы менеджеры продуктовых компаний обладали достаточной квалификацией для, по крайней мере, инициации маркетинговых исследований, которые могут показать риски при реализации продукта.

2. Развитый местный рынок, похожий на западный. Наличие местного рынка, на котором можно опробовать идеи и продукты до того, как выходить на внешние рынки является важным фактором, который сокращает риски и расходы продуктовой модели.

3. Местные венчурные инвесторы.

Если мы обратим взгляд на текущие условия российского рынка, то по всем перечисленным выше пунктам можно поставить прочерк. Прошу понять правильно, это не значит, что вообще невозможно работать по продуктовой модели. В индивидуальном порядке, на уровне компании – возможно. Но в массовом порядке – пока тяжело, хотя ситуация меняется, о чем свидетельствуют успехи таких компаний как «Яндекс», «Лаборатория Касперского» и некоторых других.

К сожалению, все, что сказано выше, в определенной степени и относится к проектной модели, только влияние негативных факторов меньше и это приводит в появлению большого количества проектных компаний. Кроме этого, негативно на проектные компании влияют следующие факторы:

1. Налоговая политика страны, которая уменьшает прибыль компании и возможность реинвестировать прибыль в развитие.

2. Несоответствующая система образования в стране, где размещено производство, что не позволяет быстро наращивать компанию.

Основной вывод

В России, на сегодняшний момент, в независимости от призывов, наибольшее развитие получает проектная модель, как более устойчивая к негативным условиям, из которых сплошь и рядом состоит Россия. На фоне проектной модели, иногда прорываются компании, которые следуют продуктовой модели. Но их достижения можно расценивать как подвиг.

Кроме этого, необходимо отметить, что существует четкая связь между компаниями, которые работают в проектной и продуктовой моделях. Например:

1. В ходе работы над проектами могут быть найдены идеи продуктов, которые могут быть разработаны за счет внутренних инвестиций компании,

потом опробованы на ряде заказчиков, а потом выделены в отдельную компанию («спин офф») и раскручены с привлечением инвестиций. В качестве похожего примера на внутреннем рынке (похожего, но не точно совпадающего) можно привести выделение электронного магазина «Озон» из проектной компании «Рексофт».

2. В ходе работы над продуктами, для повышения прибыльности компании и использовании наработанного опыта внутри продуктовой компании можно запускать проектное подразделение. В качестве примера можно привести формирование отдела заказных разработок внутри компании "Рамблер". Кстати, это иногда делается и в случае, если продукт сложен и его продажа и внедрение требует проектного подхода.

3. Творческая, научная и консалтинговые составляющие проектных компаний – необходимое условие для победы в конкурентной борьбе с индийскими, китайскими, вьетнамскими и другими юго-восточными программистскими фабриками и занятие собственной рыночной ниши отечественными фирмами.

Разработка бизнес-плана продуктовой компании

Ключевым моментом для руководства продуктовой компании и потенциальных инвесторов при принятии решения о начале финансирования разработки нового программного продукта является наличие качественного бизнес-плана данной разработки. Бизнес-план – это документ, в котором содержится информация об инвестиционном проекте. Разработка такого документа – это комплексный процесс, который требует объективной оценки всех особенностей создания и продвижения на рынок нового программного продукта.

Бизнес-план может быть использован в качестве внутреннего документа при планировании деятельности продуктовой компании и в качестве коммерческого предложения для внешней стороны – инвестора или кредитора. В последнем случае бизнес-план служит обоснованием того, что инвестор или кредитор сможет вернуть свои деньги с прибылью. В то же время бизнес-план – не только движение денежных средств, он должен отражать все составляющие инвестиционного проекта. Приведем краткие характеристики основных разделов бизнес-плана.

Резюме

Раздел должен показать потенциальным инвесторам привлекательность проекта. Главная задача – «зацепить» инвестора.

Резюме – это «концентрированное» описание инвестиционного проекта: его идеи; шагов, осуществляемых для реализации идеи; требуемых затрат; итоговых показателей. Содержание резюме должно очень четко отражать идею проекта, требуемые ресурсы, возможные риски и результаты реализации проекта в «оцифрованном» виде.

В данном разделе отражается:

• основная идея проекта;

• планируемые результаты от его реализации;

• показатели эффективности инвестиций, которые характеризуют соотношение доходов и затрат, связанных с проектом: срок окупаемости (простой, дисконтированный), внутренняя норма доходности (IRR), чистая текущая стоимость (NPV), рентабельность инвестиций (NPVR). Финансовую состоятельность проекта характеризует модель расчетного счёта проекта, график привлечения и возврата кредита (выход на точку окупаемости).

Инвесторы начинают знакомство с проектом с резюме и, нередко, рассмотрение проекта на этом разделе и заканчивается. В связи с этим необходимо отнестись к написанию резюме с особым вниманием – к четкости, понятности и содержательности его информации.

Общепринятая практика представления бизнес-плана - сопровождение бумажного варианта презентацией для инвесторов, подготовленной в Power Point. Вероятность благосклонной оценки инвестора после красивой презентации существенно увеличивается.

Информация о предприятии

В разделе необходимо четко и структурировано отразить информацию о предприятии (если проект осуществляется на действующем предприятии).

Первую часть раздела необходимо отвести для описания истории создания, области деятельности, динамики развития предприятия.

Вторая часть должна содержать анализ деятельности предприятия за предшествующий период: отражаются финансово-хозяйственные показатели работы предприятия за последний отчетный год и т.п., т.е. производится анализ финансового состояния предприятия. Если предприятие ранее привлекало кредиты, необходимо указать условия и график возврата средств.

Желательно представить информацию о финансовом состоянии предприятия не в целом за год, а в разбивке по кварталам или месяцам. Это позволит более четко представить динамику изменения финансового положения предприятия в течение года. Графики и диаграммы сделают представление информации более наглядным, а сравнение со среднеотраслевыми значениями позволят сделать вывод о конкурентоспособности предприятия.

Информация об отрасли

В этом разделе приводятся результаты анализа отрасли (развивающаяся, стабильная, стагнирующая), на которую ориентирован разрабатываемый программный продукт. Можно привести прогноз развития отрасли. Информацию можно получить в различных источниках: отраслевых министерствах, отраслевых изданиях, интернете, таможне, Госкомстате и т.п. Прогнозы развития рынка готовят также консультационные компании, когда проводят комплексные проекты.

Продукция

В разделе приводятся характеристики разрабатываемого программного продукта, приводится описание продуктов-аналогов и проводится сравнительный анализ разрабатываемого продукта и существующих аналогов (заменителей). Акцент следует сделать на преимуществах продукта по отношению к конкурентам: лучшие технические характеристики, новые функции, лучший дизайн, надежность, простота эксплуатации, более дружественный и «прозрачный» интерфейс и т.п. Для этого можно определить и проанализировать ключевые факторы принятия решения о покупке - путем проведения опросов потребителей, экспертов, целевых фокусных групп.

Производственный план

Здесь приводится краткое описание производственного процесса и технологии разработки, которое будет понятно неспециалисту, приводятся перечень работ, перечень, стоимость, условия поставки и условия оплаты необходимого оборудования, аренды помещения и каналов связи, коммунальных услуг, расходных материалов, служебных поездок и пр.

Потребители. Сбытовая политика

Доказать, что реализация разработанного программного продукта (или сервиса) не вызовет серьезных проблем – основная цель данного раздела.

В разделе должен быть приведен прогноз спроса на продукцию, география его распределения, структура, прогноз динамики развития различных сегментов, определены спросообразующие факторы и построен прогноз их динамики. Приводится характеристика основных потребителей продукции.

В разделе необходимо отразить ситуацию с патентной защитой продукции, а также ее названия. Особенно это важно при выводе на рынок нового бренда. Использование наименования продукции, которое юридически не защищено, может привести к печальным последствиям: кто-то раньше выпустил продукт или сервис с таким же именем. Рекламный бюджет проекта может «ухнуть в трубу».

Дается характеристика планируемым методам продвижения и каналам сбыта продукции. Например, создается собственная торговая сеть, или предполагается реализация по существующим каналам продаж.

Большим плюсом бизнес плана является наличие протоколов о намерениях и предварительно заключенных договоров на поставку продукции.

Ценообразование

В этом разделе дается характеристика ценообразования – методика установления цены на продукцию, рассматриваемую в проекте. Анализируется себестоимость продукции, конкурентоспособность товара по

цене, приводится прогноз продаж продукции, какова вероятность демпинга со стороны конкурентов и т.п.

Конкуренция

Оценка конкурентоспособности предприятия и разрабатываемой продукции – задача этого раздела.

В разделе приводится характеристика и анализ потенциальных конкурентов: численность персонала, структура затрат на производство, планируемые капиталовложения, объёмы продаж, система организации сбыта и производства, организационная структура и другое. Степень анализа зависит от доступности информации. Производится анализ слабых и сильных сторон конкурентов.

Определяются основные факторы конкурентоспособности предприятия: внутренние и внешние факторы конкурентоспособности.

Оценка сильных и слабых сторон продукции и услуг компании, предложения по повышению ее конкурентоспособности, предложения по корректировке продуктовой стратегии, системы сбыта и продвижения и повышению квалификации сбытового персонала – все это должен видеть в этом разделе потенциальный инвестор.

Поставщики

Приводится краткое описание поставщиков оборудования, необходимых материалов и комплектующих, аренды производственных помещений и коммуникаций связи.

При формировании плана реализации проекта желательно предусматривать «запасной выход» - альтернативные варианты снабжения производства необходимыми ресурсами. Такой подход является одним из рычагов минимизации рисков проекта. Ориентация только на одного поставщика значительно увеличивает риски реализации проекта.

	Стратегия интенсивного маркетинга	**Стратегия выборочного проникновения**	**Стратегия широкого проникновения**	**Стратегия пассивного маркетинга**
Осведомленность потребителей	Малая (или нет)		Низкая	Низкая
Емкость рынка	Большая	Малая	Большая	Средняя
Цена продукта	Средняя	Высокая	Низкая	Низкая
Конкуренция	Высокая	Незначительная	Высокая	Средняя
Затраты на продвижение	Высокие	Низкие	Высокие	Низкие

Организационный план

В этом разделе должна быть приведена характеристика системы управления проектом и описание организационной структуры предприятия.

Определяется численность персонала, планируемые затраты на оплату труда. Дается характеристика формы собственности (ООО, ОАО, др.).

Обязательно следует предоставить схематическое изображение организационной структуры предприятия и проекта.

Рабочий график реализации проекта

Тут следует описать временной график реализации проекта и проведение запланированных мероприятий. Приводится график осуществления капитальных вложений, выполнения работ, производственный график: начало производства продукции, выход на проектную мощность, объёмы производства в периоды сезонных колебаний.

Финансовый план

Раздел предназначен для определения эффективности и финансовой состоятельности проекта. Он является ключевым разделом бизнес – плана.

Составление этого раздела один из самых ответственных моментов. На основании данных финансового плана производится анализ коммерческой привлекательности проекта.

Здесь должна быть отражена информация о планируемых доходах проекта (объёмы реализации), текущих затрат проекта, об инвестиционных затратах (капитальные вложения, прирост оборотного капитала), планируемые источники финансирования, их структура (собственные, заемные), графики, условия привлечения и возврата заемных источников финансирования.

Обязательно должны быть приведены прогнозные формы финансовой отчетности: Отчет о прибыли, Отчет о движении денежных средств, Баланс.

Приводится сводная характеристика эффективности проекта. Дается краткая интерпретация показателей.

Первая группа – показатели эффективности инвестиций. Классически выделяются следующие показатели.

Срок окупаемости инвестиций (простой и дисконтированный). Рассчитывается на основании чистого денежного дохода (простого и дисконтированного). Чистый денежный поток – сумма притоков и оттоков денежных средств. Дисконтирование – приведение денежных потоков разных интервалов планирования, к начальному моменту времени (первому интервалу планирования). Суть дисконтирования заключается в том, что мы принимаем разновеликую стоимость одной условной денежной единицы в разные периоды времени.

К примеру, на рубль мы можем сегодня купить 0,1 пачки сигарет, а через год, возможно, всего лишь 0,05 пачки. Или еще. Мы точно знаем, что вложив сегодня рубль в низко рискованные ценные бумаги, или просто положив рубль на депозит в банк, мы получим через какой-то интервал времени процент с этих денег.

Так вот, дисконтирование, по сути, это учет упущенной экономической выгоды от использования наших финансовых ресурсов.

Использование дисконтирования предполагает расчёт коэффициента дисконтирования (на каждый интервал планирования), на который умножается величина чистого денежного потока каждого интервала планирования. Он рассчитывается на основании ставки сравнения.

Внутренняя норма доходности (IRR). Это показатель, который показывает, при какой ставке сравнения, чистая текущая стоимость проекта становится равной нулю. Для расчета подбираем такую ставку сравнения, при которой значение дисконтированного чистого денежного потока на протяжении периода времени, обычно срока жизни проекта, (или можно употреблять термин – срок рассмотрения проекта, звучит не так мрачно) становится равным нулю.

Чистая текущая стоимость проекта (NPV). Показывает накопленный дисконтированный доход (или убыток) проекта. Рассчитывается как сумма дисконтированного чистого потока денежных средств и остаточной стоимости проекта. Рекомендуют рассчитывать NPV без учета остаточной стоимости проекта, а уже отдельно показывать остаточную стоимость.

Остаточную стоимость можно рассчитать следующим образом.

Определяется остаточная стоимость основных средств и нематериальных активов, она может корректироваться на коэффициент «усыхания», который придется определить экспертным путем (очевидно, что актив скорее всего не удастся реализовать по остаточной стоимости). Определить коэффициент «усыхания» стоимости актива по проекту довольно сложно, особенно если оборудование специализированное, срок рассмотрения проекта большой.

Есть еще одна составляющая остаточной стоимости. Это стоимость оборотных активов – готовая продукция, интеллектуальная собственность, дебиторская задолженность, авансы поставщикам за вычетом краткосрочных обязательств (кредиторская задолженность, авансы покупателей, задолженность по заработной плате и по расчетам с бюджетом). Получаем остаточную стоимость проекта.

Вторая группа – показатели финансовой состоятельности проекта.

Это максимальная ставка процентов по кредиту и остаток свободных денежных средств на прогнозном расчете проекта.

Проводится **анализ рисков осуществления проекта**. Для этого могут быть проанализированы границы изменения основных исходных параметров проекта, при которых проект остается безубыточным.

При оценке проекта на действующем предприятии строится прогнозный финансовый план предприятия без учета реализации проекта, финансовый план предприятия с учетом реализации инвестиционного проекта.

Расчет показателей эффективности инвестиций необходимо проводить, помня о следующем:

- Расчет и интерпретация показателей эффективности полных инвестиционных вложений проводится до подбора схемы финансирования проекта. Показатели характеризуют «внутреннюю» эффективность проекта;
- Расчет и интерпретация показателей эффективности собственных средств проводится с учетом привлечения источников финансирования (собственных средств или заемного капитала).

Для составления финансового плана проекта можно воспользоваться стандартными программными продуктами.

Формирование команды разработчиков ПО

Резюме, собеседование и удерживание сотрудников

Хотя команда создаётся из отдельных людей, но вообще-то нужно работать с коллективом — обдуманно и осторожно. В этой главе мы рассмотрим основы построения коллектива: анализ резюме, собеседование с кандидатами и создание необходимых условий.

Умение отличить плохих кандидатов от хороших на основе их резюме — очень важное качество. Ничего не стоит швырнуть в корзину плохое резюме прекрасного специалиста или потратить уйму времени и сил на собеседование с человеком, с которым не стоило и встречаться.

Анализ резюме

Давайте использовать определение подходящих кандидатов из предыдущей главы применительно к анализу резюме.

• Опыт работы (квалификация)

Как давно кандидат работает в интересующей вас области и насколько сложны проекты, с которыми он имел дело? Например, имеет кандидат 2— или 10-летний опыт разработки на C++? Писал он код высокопроизводительных приложений с обработкой транзакций для финансовых институтов или простой код для вывода диалоговых окон в вузовской программке? Предыдущий опыт скажется на способности специалиста разбираться со сложными технологиями и применять их в работе. Вам нужны люди, справившиеся со сложной работой хотя бы в одной из интересующих вас областей.

• Летуны (преданность)

Один из лучших способов оценить это качество — посмотреть в резюме, не «летун» ли он: если за последние три года он сменил четыре места, его преданность работе можно поставить под вопрос. Хотя причины смены работы могут быть вполне обоснованными, такое поведение заслуживает пристального внимания.

• Предприятия, на которых кандидат работал в прошлом (отношение к делу, умение работать в команде)

Ещё одна неплохая возможность получше узнать потенциального работника — посмотреть, где он работал раньше. Вам нужны люди, которые хорошо впишутся в организацию того типа, которая у вас есть или которую вы хотите создать. Где работал кандидат: всегда только в больших компаниях или в малых? Преимущественно в отделах информационных технологий или независимых фирмах-производителях ПО? Потратил ли он большую часть времени на работу по госзаказам или три его последние работодателя были начинающими компаниями? Над чем трудился кандидат: создавал «коробочное» ПО или занимался реализацией больших проектов масштаба предприятия?

Рабочая среда, корпоративная культура, сегмент бизнеса, выбранные кандидатом в прошлом, могут многое о нём сказать — для этого даже не нужно встречаться. Так, если ваша компания начинающая, а в резюме говорится об

опыте работы с начинающими компаниями, такой кандидат может вам подойти. С другой стороны, если вы имеете дело с кандидатом, пришедшим из компании, загруженной формализованными методами и стандартами, а вы пытаетесь быстро выбросить свой продукт на рынок, вам, возможно, придётся отказать такому кандидату, особенно если есть другие факторы, указывающие на его приверженность неторопливой работе.

• «Сильные» и «слабые» глаголы (поведение)

Вы можете многое почерпнуть из того, как кандидат описывает свой предыдущий опыт. Хорошие работники обычно демонстрируют свою приверженность и ответственность за решаемые задачи и гордятся этим. Показателем такого положительного качества обычно являются «сильные» глаголы, например:

— произвёл;
— овладел;
— управлял;
— определил;
— написал;
— интегрировал;
— направил;
— создал.

Слабые, глаголы в целом означают меньшую приверженность делу и ответственность. Применение таких слов может указывать, что кандидат не в полной мере овладел предметом. Поищите такие слова:

— принимал участие;
— ознакомился;
— следовал;
— помогал;
— способствовал;
— комментировал.

• Сфера ответственности (поведение, умение работать в команде)

Оцените сферу ответственности. Насколько большим был проект? Какова была доля участия в нём кандидата? Насколько важно это было для компании? Что бы произошло при срыве проекта? Сколько людей участвовало в проекте?

• Способность письменно излагать свои мысли (умение работать в команде)

Резюме и сопроводительное письмо — первые примеры способности кандидата письменно излагать свои мысли. Они же могут быть и единственными, если вы не попросите кандидата написать что-нибудь ещё. Хорошо ли это читается? Не слишком ли многословно? Не чересчур ли сжато? Понимаете ли вы, что хотел сказать кандидат? Не забудьте оценить всё, что получили от кандидата.

• Профессиональный кругозор (жажда знаний)

Жажду знаний можно оценить (с некоторыми оговорками), исходя из широты профессионального опыта. В целом тот, кто стремится расширять свои знания, склонен принимать предложения, позволяющие ему испробовать что-то новое. Хорошей мерой профессионального кругозора кандидата является оценка его способности работать на разных уровнях абстрагирования. Может ли он

работать как с высокоуровневым кодом (например, с пользовательским интерфейсом и общей логикой программы), так и с низкоуровневыми технологиями (потоки, управление памятью, внутренняя организация ОС и т.п.)?

Резюме не даёт полной картины

Резюме лишь помогают отобрать кандидатов для собеседования. Не забывайте оценить жизненный опыт кандидата на основании сопроводительного письма и резюме. Иногда из него видны характерные черты человека, которые могут иметь значение для предлагаемой вами работы. Вполне вероятно, что умение играть на нескольких музыкальных инструментах говорит о способности работать с несколькими языками программирования, успехи в спорте — об активной натуре, а опыт военной службы — о дисциплине.

Телефонное интервью

Прежде чем назначать встречу кандидатам для собеседования, имеет смысл провести их предварительный отсев, поговорив по телефону. Телефонное интервью — эффективное средство получения дополнительных сведений о кандидатах, а также прекрасная возможность сэкономить время. Оно поможет не только понять, для какой должности в наибольшей степени подходит кандидат, но и назначать ли вообще ему собеседование. Собеседование по телефону целесообразно в следующих случаях.

• Вы хотите больше узнать о кандидате

Предварительный разговор по телефону — лучший способ решить, подходит ли вам кандидат, когда вы в нём не уверены. Резюме не даёт полного представления о кандидате, так что если у вас есть сомнения, поговорите с ним по телефону. Соберите наиболее важные сведения, которые помогут вам решить, отсеять его или назначить ему встречу.

• Нужно установить контакт, но нет возможности встретиться

Если ваш график не позволяет своевременно провести собеседование, созвонитесь с кандидатом, подтвердив при этом наличие вакансии. Так, если у вас есть вакансия разработчика пользовательского интерфейса для самых современных карманных компьютеров, сообщите об этом кандидату. Если у вас отличная команда и прекрасные условия работы, кандидат должен об этом знать. Это поможет поддерживать заинтересованность кандидата в вашей компании, когда он будет рассматривать другие предложения, а вы сможете решить, нужно ли уделять ему больше внимания, чтобы взять его себе.

Из собственного опыта

Иногда для забавы мы просматриваем старые резюме наших сотрудников. Показательно (и довольно смешно), насколько плохи их резюме в сравнении с тем, что на самом деле представляют эти люди или кем они стали. Когда-то я зачитывал такие резюме, не называя имени автора, и спрашивал у коллег, взять ли нам его на работу. Ответы были очень интересными!

Собеседование с кандидатом

Наконец появился хороший кандидат. Следующий шаг — собеседование. Далее я расскажу о принципах его проведения.

Команда, проводящая собеседование

В собеседовании должны принимать участие все сотрудники, которые будут непосредственно связаны с новичком. Не забывайте: вы создаёте команду и важно, чтобы другие её члены приняли нового сотрудника. Им будет легче это сделать, если они будут участвовать в выборе кандидатов.

Ключевые темы

При собеседовании вы должны оценить:

• квалификацию;
• преданность;
• отношение к делу;
• поведение;
• умение работать в команде:
• жажду знаний.

Оценка квалификации

На собеседовании важно оценить квалификацию кандидата. Правильней всего привлечь для этого руководителей или ведущих специалистов отдельных направлений. Это могут быть руководители отдела разработки, тестирования или разработки технической документации. Ваши специалисты должны уделить достаточно времени для понимания того, что и как кандидат уже сделал.

Вот простой совет: даже если вы, как интервьюер, не знаете предмет столь же глубоко, как кандидат, вы все равно можете задавать много вопросов для оценки его знаний. Здесь важно внимательно слушать и правильно формулировать последующие вопросы. Например:

• «Что такое потоки?»
• «Как они функционируют?»
• «Зачем они применяются?»
• «Каковы наиболее распространённые проблемы использования потоков?»
• «Опишите наиболее сложную проблему, связанную с потоками, которую вам удалось решить».

Откровенно говоря, один и тот же набор вопросов вы можете использовать для самых разнообразных тем, скажем, обсуждая COM, серверы Sun и базы данных Oracle. Если ответы становятся короче и поверхностней, стоит копнуть глубже. Но если кандидат отвечает свободно, подробно и приводит примеры из реальной жизни, вы, скорее всего, имеете дело с кандидатом, знающим своё дело.

Из собственного опыта

Как-то в NuMega мы связались с кандидатом, из резюме которого следовало, что он был архитектором сложных систем на базе COM, и мы попросили своих экспертов по COM провести с ним собеседование. Сначала кандидата попросили схематически изобразить его проект на доске, а затем

пройтись по объектной модели и объяснить проектные решения. Кандидат нарисовал на доске один квадратик, долго на него таращился, а потом сказал: «Ладно, поймали. Я соврал в своём резюме — я это не разрабатывал». Вывод: всегда нужно убедиться, что кандидат умеет делать то, что он декларирует.

Ключевые вопросы

Вопросы на собеседовании должны быть связаны с предыдущей практикой кандидата, а не с абстрактными ситуациями, для которых кандидат обычно знает «правильные» ответы. Вам нужно определить, как он работает в реальных условиях. Оценив поведение человека предыдущих ситуациях, вы можете представить, как он будет справляться с новой работой. Рассмотрим ряд общих проблем в тех областях, на которых нужно сосредоточиться. Они не претендуют на полноту — это лишь примеры вопросов, относящихся к ключевым темам.

• Квалификация.

— Опишите последний случай, когда для решения проблемы вам требовалась помощь других специалистов. Долго ли вы её ждали? Как вы взаимодействовали? Что получилось в итоге?

— Расскажите о сложной проблеме, которую вам пришлось выявлять и устранять. Что это было? Как вы её обнаружили? Каково было решение?

— Расскажите о каком-нибудь фрагменте кода, который вам нужно было написать в сжатые сроки. Как вы это делали? Получилось ли у вас и почему?

• Преданность.

— Какая часть вашего предыдущего проекта была самой сложной? Как вы к этому относились? В чём заключалась ваша роль? Что получилось в результате?

— Расскажите о ситуации, когда ваш проект должен был реагировать на внешние воздействия. Что было причиной? Какова была реакция команды? Какова была ваша реакция?

• Отношение к делу.

— Опишите последний случай, когда вы не уложились в сроки. Что случилось? Как вы реагировали? Что произошло потом?

— Опишите последний случай, когда вы оказались в затруднительном положении. Из-за чего? Как вы реагировали? Как вы вышли из положения?

• Поведение.

— Опишите последний случай, когда вы отвлеклись от своих дел, чтобы помочь кому-то другому. Почему вы это сделали? Каков был результат?

— Расскажите о ситуации, когда вы делали дополнительную работу, хотя вас об этом не просили и никто не знал, что вы это делаете.

• Умение работать в команде.

— Опишите самого сложного во взаимоотношениях человека, с которым вам приходилось работать. Что делало таким сложным общение с ним? Какова была ваша реакция? Как вы к этому относились?

— Каковы наиболее важные принципы плодотворной работы с другими людьми? Почему? Приведите примеры, подтверждающие ваше мнение.

• Жажда знаний.

— Как вы поддерживаете свои знания? Какие книги или журналы вы читаете, какие выставки посещаете?

— Опишите, что сейчас происходит на рынке с продуктом X. Что может случиться в дальнейшем?

Из собственного опыта

Один из любимых вопросов наших специалистов: «Есть ли у вас дома компьютер, пригодный для разработки?» Если у кандидата его нет, он, как правило, не интересует наших разработчиков. Они ищут фанатов своего дела, готовых потратить на него своё личное время и деньги, и писать программы бесплатно. Ищут себе подобных.

Обратная связь и завершение собеседования

Не действуйте в одиночку. Поддерживайте обратную связь с другими членами команды, проводящей собеседование. Обсуждайте с ними все «за» и «против» по данному кандидату, а также вопросы, требующие дальнейшего изучения. Не пытайтесь переубедить друг друга — ограничьтесь простым обсуждением в рамках своей компетентности.

Иногда становится совершенно очевидным, что кандидат не годится. То ли он не соответствует тому, что написано в его резюме, то ли из его заявлений вытекает, что он не вписывается в коллектив. Не бойтесь прервать собеседование, если стало ясно, что кандидат у вас работать не будет. Наше правило: если двое согласны, что лучше не продолжать собеседование, они его кончают — какой смысл попусту тратить время?

Тестирование кандидата

Проверка возможностей кандидата может дать очень наглядные результаты. Для начинающих и малых компаний тестирование, позволяющее больше узнать кандидата, чем интервью, зачастую является важным шагом, предшествующим приёму на работу. Тестирование не должно требовать большого объёма внешней вспомогательной информации. Вместе с тем тест должен быть достаточно сложным, чтобы можно было увидеть, как кандидат решает трудные проблемы.

Зачастую наиболее важная часть теста — не тест как таковой, а реакция на него. Не паникует ли кандидат? Не сдаётся ли он через пять минут? Находит ли он творческое или уникальное решение? Прилагает ли усилия? Не сломило ли его то, что он не смог закончить тест или не нашёл ответа?

Многие считают, что собеседование — порядочный стресс и без тестирования. Может быть, но работа в среде разработки ПО — стресс ещё больший. Вы должны быть уверены, что ваш человек до определённой степени умеет справляться со стрессами.

Из собственного опыта

Многие годы мы предлагаем одни и тот же программистский тест каждому разработчику. Он не требует никакого оборудования или обращения к справочникам. Кандидатов просят написать программу, отображающую на экране свой собственный исходный текст, не обращаясь к чтению файлов. Хотя для

многих кандидатов этот тест оказался сложным, он позволил многое в них открыть. Кое-кто сдавался через пару минут, другой присылал ответ потом, поскольку не нашёл решения во время собеседования. Один даже позвонил из самолёта, возвращаясь с собеседования! Не такие ли люди вам нужны?

Примеры разработок

Другой способ тестирования кандидатов — анализ примеров их разработок. Многие разработчики пользовательского интерфейса, технические писатели, специалисты по эргономике и программисты могут запросто представить вам такие образцы. Скажем, разработчики пользовательского интерфейса и эргономисты могут продемонстрировать снимки экранов поставляемых продуктов, а технические писатели — документацию или справочные файлы, над которыми они работали. У тестировщиков зачастую есть их старые планы тестирования или другая вспомогательная документация. Внимательно ознакомившись с предыдущими работами, вы сможете получить хорошее представление о предыдущем опыте и способностях кандидата. Когда вы не знаете, как поступить с кандидатом, образцы его разработок могут направить вас в нужном направлении.

Привлечение кандидата

При положительных результатах собеседования вы начинаете рассматривать кандидата как явного претендента на место. Теперь вы должны суметь объяснить, почему эта должность, эта компания и эта рабочая обстановка — именно то, что ему нужно. У хорошего кандидата предложений хватает, поэтому важно уметь кратко описать преимущества, которые он получит, работая на вас.

Чтобы привлечь кандидата, задайте себе следующие вопросы относительно проекта, группы и компании;

• что вы предлагаете в плане технологий?

• над какими продуктами будет работать кандидат?

• как можно охарактеризовать компанию, в которой будет работать кандидат?

• в чём уникальность предложения?

• какие уникальные преимущества есть у предлагаемой рабочей обстановки?

Заметьте: о зарплате и льготах в вашем предложении ни звука. О них я скоро расскажу. Вам не нужны люди, рассматривающие работу только через призму зарплаты. Прежде чем переходить к цифрам, нужно убедиться, что кандидат интересуется работой.

Из собственного опыта

В NuMega все аргументы в пользу компании абсолютно чётко определены, и все их знают наизусть. Многих кандидатов это завораживает.

• Вопрос: что вы предлагаете в плане технологий?

Ответ: Windows-программирование на нижнем уровне.

• Вопрос: над какими продуктами будет работать кандидат?

Ответ: средства разработки и отладки.

• Вопрос: как можно охарактеризовать компанию, в которой будет работать кандидат?

Ответ: стремительно развивающаяся коммерческая компания — поставщик ПО.

• Вопрос: в чём уникальность предложения?

Ответ: применение передовых закрытых технологий Microsoft и Intel; работа в элитарной команде разработчиков.

• Вопрос: каковы уникальные преимущества предлагаемой рабочей обстановки?

Ответ: непринуждённая обстановка, ориентированная на комфорт разработчиков, сочетающая работу и отдых.

Окончательное решение

Когда придёт время принять решение, не забудьте выяснить мнения каждого. Соберите всех заинтересованных лиц, чтобы обсудить их соображения и наблюдения. Это позволит выяснить одну из трёх вещей:

• вы нашли победителя;

• вы не нашли победителя;

• не ясно, кого вы нашли — мнения разделились.

Сделайте предложение кандидату, только если понятно, что вы имеете дело с победителем — не нужно лишнего риска. Если после многочисленных дискуссий у какого-то члена команды остаются серьёзные возражения, лучше остановиться. Важно учитывать мнение каждого члена команды. Особенно важно не вводить кандидата в команду насильно. Маловероятно, что команда, участвовавшая в собеседовании и процессе отбора, примет и будет помогать человеку, который ей не подходит.

Дополнительные усилия

Когда вы кончите собеседования и убедитесь, что хотите взять кандидата, может возникнуть ситуация, требующая дополнительных действий с вашей стороны. Иногда кандидаты рассматривают несколько предложений, или у них остаются какие-то сомнения или вопросы, которые не могут разрешить только специалисты по подбору кадров. Тогда вам может потребоваться человек, умеющий доводить сделки до конца, который снимет последние вопросы и сомнения колеблющегося кандидата. Он может позвонить ему домой, пригласить на ранний завтрак или поздний ужин и доверительно с ним поговорить. Само собой, тот факт, что вы уделяете особое внимание кандидату, скажет ему о том, как он для вас важен, что в свою очередь существенно увеличит ваши шансы заполучить этого кандидата.

Предложение

Когда приходит пора делать кандидату предложение, действуйте быстро и предлагайте солидные начальные условия. Нужно продемонстрировать кандидату, что он является важной частью команды. Не забывайте: вы берёте на работу только лучших и времени терять не намерены!

Я очень рекомендую предлагать солидную базовую зарплату и поощрения, стимулирующие производительность. Обычно это делается в виде премий и некоторой доли акций в соответствии с тем уровнем вклада в общее дело, который вы ожидаете от нового сотрудника.

Дальнейшие шаги

Если кандидат не принял вашего предложения сразу же, будьте готовы и дальше работать с ним. Ещё раз: у талантливых людей масса выгодных предложений. Нельзя сделать предложение и на этом успокоиться. Сделав предложение, шлите кандидату электронную почту, письма, звоните. Не теряйте с ним контакт на этой критической стадии. Если поступили новые предложения, возникли другие вопросы или осложнения, вы должны о них знать. Если вы не получаете ответа ещё несколько дней — все равно продолжайте и доведите дело до конца.

Если вы упустили кандидата

Принятие решения — процесс эмоциональный, здесь и мелочи могут сыграть определённую роль. Когда кандидат говорит «нет», убедитесь, что на то есть серьёзные причины, а не какие-то легко разрешаемые пустяки. Для этого нужно разобраться, почему предложение отклонено. Не успокаивайтесь, услышав о «лучшем предложении». Нужно понять, что лежит за этим ответом.

Из собственного опыта

У нас как-то был хороший кандидат на должность тестировщика, отказавшийся от нашего предложения по «семейным обстоятельствам». Поговорив с ним, мы выяснили, что он обещал свозить свою семью в диснеевский парк. Он думал, что у него не будет свободного времени из-за жёсткого графика, связанного с выпуском нашего продукта. Вообще-то он был прав — нам были нужны люди, готовые сразу приступить к работе и довести её до конца. Но, узнав причину отказа, мы обсудили ситуацию с руководством и согласились, что он — ценное дополнение к нашей команде и достоин того, чтобы учесть его «семейные обстоятельства»

В другом случае мы сделали предложение прекрасному разработчику, который затем решил продолжать работу над своим текущим проектом, так как в нём была нужда и он должен был с этим примириться. Мы, конечно, огорчились, но оценили его позицию и выразили надежду, что он свяжется с нами, завершив проект. Мы не пытались обсуждать с ним его обязательства, нельзя убеждать человека, чтобы он отказался от тех ценностей, ради которых мы его сами принимаем на работу. Вдруг, оказавшись в затруднительном положении, он снова нам позвонил, и мы взяли его на работе. Мы готовы были его ждать, уверенные, что он будет так же предан нам, как бывшему работодателю.

Удерживание сотрудников

Набрав людей, важно удержать их компании. Есть три основные группы причин, по которым люди остаются у своего работодателя.

• Профессиональные

Хорошие работники любят работать. Они очень гордятся своим делом и хотели бы видеть в нём что-то значительное. Люди должны знать, что их работа важна и по достоинству оценивается. Им нужно знать: их компания выделяется среди других и они внесли свою лепту её успех. Важно отдавать себе отчёт в наличии таких потребностей и обеспечить обратную связь как с отдельными людьми, так и с группами.

С другой стороны, даже если у вас важнейший проект в мире, надо время от времени давать сотрудникам возможность заниматься новыми вещами. Новые интересные задачи, новые коллеги и технологии, с которыми они столкнутся на новом месте, не позволят им потерять интерес к работе.

• Финансовые

Исследования показывают, что деньги — не главная причина смены работы. Хотя, конечно, если у вас работают суперпрофессионалы, платить им нужно хорошо. Оплата должна включать базовую зарплату, премии за особые достижения и периодические выплаты, стимулирующие заинтересованность в долговременном успехе компании. Такая схема оплаты не даёт сотрудникам расслабляться и удерживает их, если акции компании растут.

Талантливые люди редко испытывают трудности с поиском высокооплачиваемой работы. Но если вы предлагаете больше среднего, люди считают, что вы их цените высоко, и даже более солидная зарплата в других местах становится для них не такой привлекательной. Риск потерять человека из-за денег будет меньше, если вы и другими способами демонстрируете, что вы его цените. Если же его интересуют только деньги, это, вероятно, не лучший выбор.

• Социальные

Рабочее место, являющееся частью социальной среды, может чудесным образом влиять на удерживание сотрудника. Если люди общаются со своими коллегами и довольны рабочей обстановкой, очень маловероятно, что они захотят поменять работу (если при этом удовлетворены их профессиональные и материальные потребности). Нельзя недооценивать удобство офиса и мощность множества компьютеров, находящихся в распоряжении каждого члена коллектива. Добавьте сюда заботу о здоровье и отдыхе, и у вас — рецепт сплочённого коллектива.

Методики удерживания сотрудников

Лучший способ сохранить работников — уделять одинаковое внимание всем трём рассмотренным сферам. Хотя легче достичь превосходства в какой-то одной области, реальные выгоды компания получит, обеспечив баланс между всеми тремя. Вот некоторые рекомендации, как достичь такого баланса.

• Профессиональная сфера.

— Убедитесь, что люди получают новые навыки и пробуют что-то новое.

— Люди должны знать, что они отвечают за свою работу.

— Люди должны осознавать важность своих продуктов и проектов.

— Хвалите людей как лично, так и публично.

— Чаще переводите людей из одной группы в другую, чтобы обеспечить их рост и взаимозаменяемость.

— Узнавайте о целях карьеры своих сотрудников и обеспечивайте их карьерный рост.

• Финансовые условия.

— Зарплата и другие выплаты талантливым сотрудникам должны быть выше средних по отрасли.

— Выдавайте премии за выдающиеся достижения.

— Премируйте ведущих сотрудников акциями компании.

• Социальная сфера.

— Прикрепляйте к новичкам старых сотрудников, чтобы они по-дружески помогали им прижиться в коллективе.

— Организуйте внеурочные мероприятия (спортивные игры, походы в кино).

— Заботьтесь о социальных контактах между членами коллектива.

— Поощряйте социальную активность как на рабочем месте, так и вне его, не рассчитывайте лишь на вечеринки по большим праздникам!

Типичные проблемы и их решение

Далее мы обсудим ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Собеседование: проблемы и решения

• Слабые методики и подходы

Это очень распространённая проблема. Большинство интервьюеров или новички, или плохо справляются со своим делом. Люди, проводящие собеседование с кандидатами, должны иметь соответствующие знания и быть обучены (хоть неформально) — они должны знать, что и как делать. Если сотрудник оценивает кандидатов просто по интуиции и лишь пожимает плечами, когда его просят обосновать своё решение, вы кончите тем, что будете нанимать не тех людей и упускать достойных кандидатов.

• Плохо описанная вакансия

Собеседование должно проводиться для конкретной вакансии (если только это не ознакомительное собеседование). Если вы не знаете функциональных обязанностей вакантного поста, вы получите совершенно разные мнения членов группы, проводящей собеседование, относительно приемлемости кандидата. Чёткое описание работ для всех вакансий должно быть доступно всем интервьюерам задолго до прихода кандидата

• Слишком много времени уделяется неквалифицированным кандидатам

Не тратьте на кандидата больше времени, чем нужно. Совершенно нормально дать знать кандидату, что он не подходит. Продумайте способ досрочного завершения собеседования. Не тратьте своего времени и времени кандидата на собеседование, которое не приведёт к положительным результатам.

• Чрезмерная или недостаточная реклама собственных возможностей

Некоторые организации не предоставляют о себе достаточных сведений. Кандидата интервьюируют так дотошно, что у него не остаётся возможности узнать о вакансии. У него остаётся масса вопросов, и он не знает, подходит ли ему эта вакансия. Другие организации слишком много себя рекламируют и по сути не интервьюируют кандидата. Вы должны знать сильные стороны компании и рассказать о них в подходящий момент, но не в процессе собеседования.

• Нереализованные возможности

Вам нужно иметь пару людей, знающих, как довести работу с кандидатом до конца. Кандидат не должен уйти, не получив ответа на важные вопросы или не совсем понимая, что ему предлагается. У вас должен быть ответственный сотрудник, который может ответить на все вопросы, если кандидат не принял предложения. Он должен ориентироваться в широком круге проблем, включая видение будущего компании, возможности роста по службе и уметь сравнивать и противопоставлять альтернативные предложения.

• Медлительность

Хороших кандидатов трудно найти. Если вы уверены, что нашли достойного, — действуйте быстро. Не стесняйтесь пригласить кандидата на интервью вечером или предложить прилететь на выходные. Будьте готовы провести собеседование в тот же день. Я не предлагаю спешить с собеседованием, но бывают обстоятельства, требующие быстрого принятия решения, в том числе во внеурочное время.

Дело нужно поставить так, чтобы вы могли сразу принять кандидата на работу. Время исключительно важно при поиске талантливых людей, и почти всегда вам придётся конкурировать с другими компаниями. Нет ничего неприятнее, чем найти прекрасного кандидата, а потом услышать, что он принял другое предложение, пока вы раскачивались.

Сохранение сотрудников: проблемы и решения

• Неправильный баланс

Большинство проблем с удерживанием сотрудников связано с неправильным балансом профессиональных, финансовых и социальных факторов. Вы не сможете долго жертвовать чем-то одним в пользу другого. Нужно на регулярной основе обеспечивать баланс между профессиональной удовлетворённостью, денежным стимулом и социальной поддержкой. Не ждите, когда начнут возникать проблемы.

• Текучесть кадров

Текучка существует всегда. Меняются личные обстоятельства и приоритеты. Люди уходят по причинам, которые вы не можете контролировать: родился ребёнок, нужно быть поближе к родным, далеко до работы… С другой стороны, текучка может указывать на серьёзные проблемы в коллективе или организации. Чтобы оставаться в курсе причин текучести кадров, беседуйте с людьми перед их увольнением и прислушивайтесь к их замечаниям. Если обнаруживается внутренняя проблема, спросите других сотрудников, разделяют ли они такую точку зрения. В больших организациях неплохо вести список

причин увольнения сотрудников. Эти сведения помогут отслеживать тенденции и принимать соответствующие меры.

Организация проекта

Как бы ни были талантливы люди, они всё равно не смогут работать с максимальной эффективностью, если их не организовать правильно. Проекты часто страдают от недостатка организованности и неясностей в распределении ролей и обязанностей. Каждый должен знать свой манёвр в общем контексте проекта.

В этой главе мы подробно разберём модель организационной структуры, используемой в NuMega, а также рассмотрим роли, обязанности и навыки, необходимые участникам группы в рамках этой модели.

Модель организационной структуры компании

Программы, как правило, создаются коллективами, а не одиночками. Команда разработчиков — это группа людей с различными техническими навыками, работающих над реализацией общего проекта. Поскольку разработать ПО довольно сложно, в команде требуются специалисты с самыми разными навыками и способностями, необходимыми для создания продукта. Вот какие специалисты должны быть в группе:

• основной состав группы — специалисты, полностью занятые в создании нового программного продукта:

— менеджеры проекта;

— программисты;

— тестировщики;

— разработчики документации;

— инженерные психологи;

— технологи по разработке ПО;

• вспомогательная группа — специалисты, не занимающиеся созданием программ, но, тем не менее, играющие важную роль в реализации проекта:

— группа менеджмента и маркетинга продукта;

— специалисты по технической поддержке ПО;

— администраторы бета-тестирования.

Очень важно, чтобы перечисленные функциональные подразделения участвовали в работе над проектом с самого начала. Чем раньше люди смогут понять суть требований к продукту и принять участие в их критическом анализе, тем лучше подготовятся к исполнению собственной миссии и ощутят свой вклад в успех проекта. Кроме того, чтобы завершить создание продукта в срок, все перечисленные подразделения должны работать параллельно на протяжении всего цикла разработки. Решение этой задачи будет описано в главе 11 — там мы рассмотрим включение в график проекта взаимно скоординированных во времени промежуточных этапов.

С другой стороны, если при подборе кадров какие-либо функциональные подразделения будут не (недо-) укомплектованы, то реализовать такие важные условия разработки ПО, как глубокое понимание задач, синергизм в работе и постепенный прогресс, будет невозможно. Я не настаиваю на том, чтобы все

подразделения были полностью укомплектованы к первому дню работы над проектом, но по крайней мере их представители (хочется надеяться, что это будут ведущие специалисты) должны работать над проектом с самого начала. Нельзя недооценивать как важность этого требования, так и трудность его реализации.

Управление проектом

В качестве примера структуры оптимальной системы управления небольшой компании я подробно остановлюсь на организации управления в NuMega, позволившей ей справиться с трудностями. Как и любой молодой компании (и большинству начинающих групп разработчиков), нам требовалось выполнить большую работу в сжатые сроки, при этом ресурсы были сильно ограничены. Мы знали: чтобы воспользоваться всеми преимуществами талантливых сотрудников, которых нам удалось привлечь с таким трудом, необходимо их эффективно организовать. Нужна такая структура организации, которая позволила бы оперативно реагировать на возникающие трудности, сводить к минимуму разного рода издержки и которую можно было бы расширить в дальнейшем. Чтобы реализовать сформулированные требования, мы решили задействовать простую модель структуры организации, в которой за все аспекты разработки продукта отвечает один менеджер проекта. В сферу его ответственности входит наблюдение за всеми программистами, тестировщиками, технологами и разработчиками документации, т.е. за основным составом группы. Важнее всего, что все способные сотрудники были собраны под началом одного менеджера.

Остальные сотрудники (группа технической поддержки, администраторы бета-тестирования, группа менеджмента и маркетинга продукта) не отчитывались перед менеджером проекта, но работали с ним в прямом контакте для решения своих проблем и получения всего необходимого для работы. Этот подход дал неплохие результаты, так как приоритетной обязанностью каждого из вспомогательных подразделений было решение конкретной задачи (анализ рынка, формирование ценовой политики, обработка входящих сообщений, реклама и т.д.) и не предполагало повседневного участия в разработке продукта. Поскольку же все подчинялись одному менеджеру проекта, взаимодействие функциональных подразделений стало заметно проще и понятнее.

Хотя избранная нами структура организации работала хорошо, применение следующих принципов способствовало дальнейшему повышению её эффективности.

• Гибкое использование ресурсов

Менеджер проекта мог выделять нужные ресурсы и направлять группу специалистов для решения любой отдельной проблемы, устранения той или иной неполадки или поддержания какой-либо инициативы. Такая система позволила менеджеру проекта распределять ресурсы в соответствии с текущими внутренними приоритетами проекта и обеспечила полноту использования и оперативную балансировку ресурсов согласно быстро меняющимся потребностям проекта.

• Ответственность за распределение специализированных ресурсов

Все ресурсы находились в руках его менеджера, а команда в полном составе работала над проектом с первого и до последнего дня. Таким образом, была группа людей с единым набором приоритетов, работавших над решением единой задачи и под руководством одного человека. Такая структура позволяла привлечь каждого сотрудника к непосредственному участию в проекте уже на начальных этапах работы, в результате каждый в большей мере испытывал чувство ответственности и причастности к достигнутым результатам. Люди лучше представляли себе все особенности и ограничения проекта, а также причины тех или иных решений, что позволяло лучше спланировать проект и организовать тестирование, а также обеспечить проект более качественной документацией.

• Централизованное принятие решений

Поскольку проект целиком находится в ведении одного менеджера, он может оперативно принимать критически важные решения, разрубая «Гордиевы узлы», когда не удалось достичь согласия.

• Более чёткое взаимодействие

Каждый, у кого возникают вопросы или трудности, может обсудить их с менеджером проекта. Таким образом, простая и понятная схема взаимодействия участников позволяла эффективно устранять затруднения, возникающие у участников как основной, так и вспомогательной групп.

• Инициативная ответственность

Менеджер проекта — это не просто управляющий, но один из тех, кто заинтересован в успехе продукта. Он должен быть в курсе конъюнктуры и тенденций рынка, а также чётко представлять ценность функций программы. Без этих знаний он не сможет оперативно оценивать ход реализации ПО и обеспечить выполнение работы на должном уровне. Менеджер проекта работает в одной упряжке с менеджером продукта, формулирующим требования рынка и курирующим экономические аспекты создаваемого продукта. Оба вносят свой вклад во всех областях: в формирование политики лицензирования, ценообразование, продвижение и сбыт продукта. В конечном счёте они вместе отвечают за успех продукта и наделены полномочиями принимать ключевые решения. Обладая большой властью, они могут быстро принимать нужные решения. В то же время участники команды, зная, что они работают непосредственно с теми, кто принимает решения, в курсе их идей и уверены в том, что их работа не просто нужна, но имеет решающее значение для успеха проекта.

Из опыта

Обычно в NuMega самое важное собрание, посвящённое проекту, — вводное. Всем участникам проекта, собравшимся в одной комнате, говорили, что продукт может быть создан только с участием каждого из них — все должны посвятить себя реализации проекта. Пока проект не будет отправлен заказчику, каждый должен выполнять свои обязанности. Если проект будет успешным и заказчик его одобрит, это будет результатом общих усилий команды, а если проект будет сорван — опять же в этом будут виноваты все. Методика, в рамках

которой команда получала прямые полномочия, позволяющие ей принимать большинство решений, необходимых для работы, приносила восхитительные результаты.

Ведущие специалисты

Мы также назначили ведущих специалистов в каждом из функциональных подразделений. Ведущий специалист является экспертом, возглавляющим работу во вверенной ему области. На протяжении всего цикла он очень тесно сотрудничает с менеджером и ведущими специалистами других подразделений. Ведущие специалисты играют важную руководящую роль, управляя разрешением проблем и принятием решений во вверенных им ключевых областях. Такая система позволила без труда увеличить число сотрудников функциональных подразделений, поскольку их возглавлял человек, управляющий созданием данной части продукта. Связи между менеджером проекта и ведущими специалистами функциональных подразделений показаны ниже (рис. 3-1).

Рис.3-1. Связи между менеджером проекта и ведущими специалистами функциональных подразделений.

Поскольку мы нанимали высококвалифицированных и опытных специалистов, они могли работать почти без надзора менеджера. Их нужно было лишь познакомить с обязанностями и планом. Важнее всего, что такая модель позволила держать ключевые ресурсы под контролем одного человека — менеджера, который непрерывно был в курсе повседневной работы группы.

Роли и обязанности

Рассмотрим основные роли и обязанности лидеров функциональных подразделений, участвующих в создании продукта.

Менеджер проекта

Совершенно ясно, что менеджер играет ключевую роль в реализации проекта. Его функционал включает следующие многочисленные роли и обязанности.

• Подбор кадров и управление ими

Менеджер проекта отвечает за создание команды и управление её составом. Он также курирует кадровое обеспечение, планирование карьеры, кадровую политику, анализ и проверку работы сотрудников и поддержание «боевого духа» группы. В его обязанность также входит найм новых сотрудников, повышение мастерства и развитие навыков специалистов, а также поддержание мотивации и целенаправленной деятельности людей во время работы над проектом.

• Формулирование и исполнение плана проекта

Формулированием и исполнением плана проекта руководит менеджер проекта. Он подбирает группу специалистов, формулирующих и согласовывающих требования, а также отслеживающих их выполнение. Когда список требований готов, менеджер составляет план, регламентирующий все действия, необходимые для реализации проекта: программирование, тестирование, разработку документации, работу технологов по разработке ПО и

инженерных психологов. План также учитывает другие ключевые аспекты создания продукта, а именно: составление графика работы, реализацию технологии программирования, подбор кадров, а также неопределённости и риск. Это не значит, что все решения принимает только менеджер проекта, но именно он отвечает за то, чтобы довести создание всех частей продукта до конца во взаимодействии с ключевыми участниками проекта, и обязан донести общий план до каждого участника группы. Подробнее все эти вопросы мы рассмотрим во второй части книги.

При реализации функций ПО во время разработки постоянно приходится идти на компромисс. Именно менеджер проекта отвечает за то, чтобы соответствующие решения принимались своевременно и были согласованы с командой, которая должна быть в курсе принятых решений. Если корректировки приведут к коренным переменам в направленности продукта, он должен довести решение и все его последствия до сведения каждого, кого оно затрагивает.

Поскольку менеджер проекта не только отвечает за реализацию функций продукта, кадровое обеспечение и балансировку ресурсов проекта, но и курирует график реализации всего проекта, именно он отвечает за соблюдение графика работы над проектом и обязан вносить соответствующие коррективы в случае возникновения проблем.

Менеджер проекта создаёт главный график работ на основе сведений, поданных всеми участниками проекта. Как правило, в соответствии с этим графиком работа над проектом подразделяется на ряд промежуточных этапов и базовых уровней. Менеджер проекта должен непрерывно следить за исполнением графика, вносить нужные изменения и сообщать о них группам разработчиков и другим группам, в сотрудничестве с которыми ведётся работа (группы менеджмента, технической поддержки и администраторов бета-тестирования), а также верхнего эшелона управления.

• Руководство командой

Менеджер проекта отвечает за плавное и эффективное исполнение разработки продукта. Менеджер проекта устраняет возникающие препятствия и обеспечивает всё необходимое для успешной работы команды. Он должен определять проблемные области, работать над ускорением решения проблем и поддерживать команду в состоянии сосредоточенности и гармонии. Он также должен быть готов выступить в роли инструктора или наставника, обладающего достаточными знаниями и опытом в разных областях, чтобы оценить успехи команды и при необходимости помочь ей.

• Обеспечение связи между подразделениями

Менеджер проекта — главное связующее звено между разработчиками и группой менеджмента и маркетинга. Он отвечает за сбор пожеланий в этих группах и воплощение их в плане проекта. Во время выполнения проекта он также отвечает за доведение возникших трудностей или изменений в плане проекта до сведения менеджера продукта и менеджера по маркетингу. Кроме того, проанализировав планы менеджмента и маркетинга, менеджер продукта должен дать свой отзыв о них.

Менеджер проекта также является главным каналом связи между группами разработчиков, технической поддержки и администраторов бета-тестирования. Он должен решать любые критические проблемы, возникающие как во время тестирования продукта клиентами (бета-тестирования), так и после выпуска;

• Обеспечение готовности продукта

Менеджер проекта также отвечает за создание максимально завершённого и качественного продукта. Таким образом, ответственность за достижение всех целей, поставленных перед программистами, разработчиками документации и инженерными психологами ложится в конечном счёте на плечи менеджера проекта.

Программисты

Можно выделить три основных категории технических специалистов: ведущий разработчик (программист), ведущий программист, отвечающий за реализацию определённой функции и рядовой программист (рис. 3-2).

Рис. 3-2. Связи между ведущим разработчиком, ведущими программистами, ответственными за реализацию определённых функций ПО, и рядовыми программистами.

Ведущий разработчик

Это главный специалист по разработке ПО. Эту должность, как правило, занимает один человек. Поскольку он играет ключевую роль в разработке ПО, занимающий эту должность специалист должен быть достаточно зрелым и квалифицированным, чтобы справиться со сложными техническими и кадровыми проблемами, постоянно возникающими во время цикла разработки. В число его обязанностей входит:

• наблюдение за соблюдением архитектурных и технических спецификаций продукта;

• подбор ключевых технологических инструментов и стандартов;

• диагностика и разрешение всех технических проблем;

• выполнение роли технического инструктора и консультанта для участников проекта;

• наблюдение и контроль за работой групп разработчиков документации, тестировщиков и технологов;

• мониторинг состояния (ведение списка обнаруженных ошибок);

• подбор инструментов разработки, метрик и стандартов и наблюдение за их использованием;

• ну и, конечно, программирование, программирование и ещё раз программирование.

Ведущие программисты, отвечающие за реализацию отдельных функций

Отвечают за реализацию отдельных функций продукта, часто на основе конкретной технологии. Обычно определение функций формулируют довольно широко, например, «интеграция с IDE» или «разработка API доступа к БД». Обязанностями ведущих программистов, отвечающих за создание отдельных функций ПО, являются:

• согласование архитектурных вопросов с коллегами, ответственными за разработку других функций;

• формулирование требований к функциям и их критический анализ;

• проектирование функций;

• снабжение тестировщиков и разработчиков документации техническими материалами;

• ну и, конечно, программирование, программирование и ещё раз программирование.

Рядовые программисты

Работают над реализацией определённой функции ПО обычно под руководством ведущего программиста, ответственного за эту функцию. Они отвечают за реализацию конкретных аспектов этой функции, например, за «интеграцию в IDE окон X, Y и Z» или «написание для API баз данных методов create, update и delete». В круг их обязанностей входит:

• реализация функции;

• её тестирование;

• исправление ошибок в реализованной функции;

• помощь техническим писателям в документировании реализованной функции;

• помощь тестировщикам в испытаниях этой функции.

Тестировщики

Отвечают за составление и исполнение плана тестирования программы, создаваемой в рамках проекта. Чтобы обеспечить истинное партнёрство между теми, кто пишет код и теми, кто его тестирует, роли и обязанности группы тестировщиков должны быть «параллельны» обязанностям разработчиков.

Традиционно группы тестировщиков и разработчиков функционируют раздельно, обладая независимыми полномочиями в отношении качества ПО, иначе велика вероятность того, что события пойдут по сценарию хорошо известной сказки про лису, которой доверили охранять курятник. С другой стороны, наличие группы тестировщиков со своим менеджером, обладающим равными полномочиями с менеджером проекта, может привести к конфронтации. Со временем группы могут отдалиться друг от друга, и между ними могут возникнуть натянутые отношения, отравляющие любые начинания.

В NuMega удалось избежать обеих проблем, передав право окончательного решения вопросов о качестве ПО в руки менеджера проекта. Он должен предоставить качественный продукт, и именно с него спросят за любые проблемы с продуктом. Принимая решение о готовности продукта, ему приходится полагаться на результаты испытаний, проведённых группой тестировщиков. Такая структура организации (рис. 3-3) позволяет группе тестировщиков оставаться независимой, так как она является самостоятельным подразделением под руководством своего ведущего специалиста. Однако, будучи подотчётными тому же менеджеру, что и разработчики, они ощущают, что их воспринимают так же, как любых других участников группы, и обращаются с ними соответственно.

Рис. 3-3. Связи между группами разработчиков и тестировщиков.

Ведущий тестировщик

Отвечает за организацию и исполнение тестирования ПО в период разработки. Он сам должен обладать хорошими навыками тестирования и быть способен возглавить других тестировщиков и направить их усилия в нужное русло. Его обязанности таковы:

• Составление плана тестирования продукта

План тестирования регламентирует работы по испытанию программы, т.е, что, как и когда будет протестировано. Ведущий тестировщик также занимается решением дополнительных проблем, обеспечением возникающих потребностей и необходимых ресурсов.

• Исполнение плана тестирования

Ведущий тестировщик отвечает за исполнение плана тестирования на протяжении всего цикла разработки. Он сравнивает результаты тестирования продукта со спецификациями базовых уровней и промежуточных этапов, определённых в графике разработки продукта, а также следит за тем, чтобы тестирование новых функций программы проводилось своевременно.

• Автоматизация испытаний

Управление автоматизацией наиболее критических тестов согласно плану с целью ускорить тестирование. Испытание готовит и проводит группа, но ответственность за проведение испытания лежит на ведущем тестировщике.

• Проведение регрессивного тестирования

Ведущий тестировщик следит, чтобы после каждой сборки программы проводилось её регрессивное тестирование. Лучше проводить эти тесты (известные также как базовые тесты) ночью, чтобы их результаты были готовы к утру, Ведущий тестировщик отвечает за ежедневный анализ результатов и регистрацию обнаруженных ошибок в системе слежения за ошибками.

• Выбор инструментов, метрик и стандартов для тестирования

Во время реализации проекта ведущий тестировщик отвечает за выбор и использование инструментов, метрик и стандартов для тестирования, т.е, делает то же, что и ведущий разработчик для своей группы. Так, ведущий тестировщик отвечает за целостность данных в системе слежения за ошибками аналогично тому, как ведущий разработчик отвечает за целостность данных в системе управления исходным текстом.

Инженер по автоматизации

В основном занимается созданием автоматизированных тестовых заданий согласно плану. Этот специалист, как правило, обладает большим навыком работы с инструментами для автоматизации тестирования, написания сценариев и часто программирования. Перед инженерами ставится задача по автоматизации тестирования набора функций программы, и они концентрируются на тестировании некоторых частей продукта, работу которых можно описать количественно. Это позволяет им тесно сотрудничать с ведущими специалистами, отвечающими за разработку этих функций. Круг обязанностей инженера по автоматизации более узкий в сравнению с другими участниками группы, так как он должен обеспечить автоматизацию тестирования той или иной функции лишь после завершения программирования. К его обязанностям относятся:

• планирование испытаний;
• автоматизация испытаний;
• оценка и выбор инструментальных средств.

Рядовой тестировщик

Отвечает за исполнение плана тестирования, составленного ведущим тестировщиком. Обычно тестировщику приходится играть роль пользователя программы, и он должен знать её функции, как свои пять пальцев. Он должен быть посвящён во все секреты конструкции программы и быть способным провести тестирование пользовательского интерфейса для его подгонки и шлифовки. В круг основных обязанностей этого специалиста входит:

• тестирование программы установки, всех функций и пользовательского интерфейса согласно плану тестирования;
• проведение автоматизированных испытаний;
• регистрация результатов автоматизированных испытаний и анализ обнаруженных неполадок;
• окончательное подтверждение устранения ошибки;
• подготовка среды для испытаний.

Группа разработчиков пользовательской документации

Обеспечивает пользователя справочными материалами: печатной документацией, электронной справочной системой, обучающими программами и карточками быстрой справки.

Ведущий разработчик пользовательской документации

Отвечает за составление плана создания документации для всего проекта. Опираясь на своё знание продукта и потребностей пользователей и принимая в расчёт доступные ресурсы, он составляет план, регламентирующий виды и сроки создаваемой документации.

Ведущий разработчик пользовательской документации отвечает за определение стандартов документации (и участвует в их создании) и следит, чтобы в продукте были отражены самые последние изменения в правилах и технологиях написания технической документации.

Рядовой разработчик пользовательской документации

Помимо написания и производства документации, группа разработчиков пользовательской документации отвечает за удобство в работе и качество ПО. Часто недостатки ПО заметны прежде всего именно техническому писателю, так как, работая с продуктом, ему приходится ставить себя на место пользователя. Приведу пару примеров:

• Поскольку разработчик пользовательской документации создаёт руководство, описывающее использование продукта, часто именно он первым обнаруживает несогласованности, проблемы с функционированием продукта и недостатки в реализации функций. Нет ничего необычного, когда технический писатель заявляет: «Да, на собрании, посвящённому анализу спецификаций, у меня никаких вопросов не возникло. Но теперь, когда я начал писать руководство пользователя, мне ясно, что пользователю придётся выполнить целых десять действий, чтобы решить эту задачу, — чепуха какая-то!» Таким образом, он на ранних стадиях цикла разработки выполняет весьма ценную параллельную

проверку удобства использования программы и даёт отзывы, позволяющие скорректировать недочёты.

• Разработчик пользовательской документации должен работать с программой практически ежедневно, чтобы точно задокументировать новые функции и идти в ногу с изменениями, вносимыми в программу. Регулярная работа с продуктом позволяет обнаружить проблемы с качеством на ранних стадиях цикла разработки, когда решать их ещё не так трудно. Хотя разработчики документации не могут заменить тестировщиков, они пытаются работать с фрагментами программы, собранными вместе, поэтому они могут обнаружить ряд важных ошибок, которые в противном случае всплывут гораздо позже. В этом смысле разработчик документации проводит дополнительную проверку качества продукта и часто даёт весьма реалистичную оценку его качества.

Инженерные психологи

Впечатление, которое оставит продукт у пользователя, критически важно для его успеха на рынке. Интерфейс, документация, упаковка — всё должно работать на то, чтобы создать у клиента положительное впечатление о продукте.

Мы в NuMega всегда были убеждены, что именно первые 20 минут общения с нашим продуктом определяют, примет ли его пользователь и будет ли продолжать с ним работать. Это явление получило название «первоначальное впечатление от работы с продуктом». Если продукт не оставил у пользователя положительного впечатления и не помог ему легко и быстро решить свои проблемы, маловероятно, что этот продукт будет регулярно использоваться или будет по-настоящему ценным для потребителя.

Инженерные психологи помогают справиться с этими проблемами. Ведущий специалист по инженерной психологии отвечает за перевод требований к проекту в фундаментальные задачи, которые должен решать пользователь, и далее в модель пользовательского интерфейса. Эти факторы оказались весьма существенными для организации оптимизации и определения других приоритетных направлений работы команды. Так, тестировщики концентрируют свои усилия на проверке ключевых задач, определённых группой инженерных психологов, а разработчики документации будут следить за тем, чтобы этим задачам было уделено наибольшее внимание в учебниках и руководстве пользователя. Эти задачи, определяющие основную ценность предлагаемого продукта, непременно нужно завершить в срок и выделить для этого достаточно времени.

Этот момент имеет решающее значение: все участники группы должны знать, какие задачи наиболее важны для пользователя и как они должны быть реализованы в программе. Если кому-то в группе эти задачи будут неизвестны, вся группа рискует погрязнуть в бессмысленной работе. Приходилось ли вам видеть, как разработчики и тестировщики корпят над явно второстепенной функцией, когда главные функции программы работают плохо или вовсе не работают; или группы, завязшие в бесконечных спорах и конфликтах о пользовательском интерфейсе на завершающих этапах бета-тестирования? Скорее всего, в таких группах отсутствует единое понимание приоритетных

потребностей клиента, и способ их реализации там никогда заранее не обговаривали.

Специалист по инженерной психологии должен:

• транслировать формулировки требований в ключевые задачи;

• разрабатывать дизайн пользовательского интерфейса (макеты диалоговых окон и т.д.) для решения этих задач;

• тестировать разработанный дизайн и согласовывать его с командой;

• определять, как сформировать положительное первоначальное впечатление от продукта;

• проводить подгонку и доводку пользовательского интерфейса;

• работать с заказчиком после выпуска ПО.

Технологи по разработке ПО

Обеспечивают работу базовых служб, необходимых для поддержания работоспособности принятой модели разработки ПО. Эту работу должны выполнять соответствующие специалисты или сами разработчики, даже если для этого придётся продлить календарный план. Не впадайте в самообман, думая обойтись без этой работы: выпустить ПО вовремя без неё не получится. Подробнее о работе технологов по разработке ПО см. главу 7.

В общем случае у технолога по созданию ПО три основные обязанности:

• Создание и сопровождение подходящей среды для сборки продукта

Сборка программы — необходимый первый шаг, который должен быть завершён как можно раньше. Ежедневная сборка программы — ключ к успеху проекта, без неё невозможно воплотить многие концепции, изложенные в этой книге.

• Создание и сопровождение процедуры установки

Чтобы извлечь все выгоды частой сборки программы, нужно, чтобы каждая новая сборка устанавливалась автоматически. Кроме того, установочную процедуру требуется сопровождать и обновлять по ходу цикла разработки, а также следить, чтобы для выполнения этой задачи было выделено достаточно ресурсов.

• Сопровождение и администрирование систем управления исходным текстом

Важно, чтобы за сопровождение системы управления исходным текстом постоянно отвечал один и то же специалист. Об инструментах для управления исходным текстом см. главу 4.

Группа менеджмента и маркетинга продукта

С точки зрения технических подразделений, эта группа играет две роли; первая — сбор информации, а вторая — её выдача.

В рамках первой роли группа менеджмента и маркетинга продукта определяет приоритетные сегменты рынка, экономические и потребительские требования к продукту. Наличие у разработчиков ясных, сжатых, реальных и обоснованных экономических требований к продукту имеет решающее значение для его успеха. В дополнение к сбору сведений о необходимой функциональности продукта в обязанности этой группы входит формулирование требований к

комплектности продукта, установочной программе, лицензированию и документированию.

С другой стороны, группа менеджмента и маркетинга помогает техническим подразделения представить продукт за пределами компании. Сюда относится реклама, обучение и оснащение необходимыми средствами специалистов по сбыту, аналитические совещания, пресс-релизы и брифинги, а также новости, публикуемые в Web.

Технические группы полагаются на группу менеджмента и маркетинга в распространении информации о своей работе; поддержка этой группы необходима, чтобы начать проект и обеспечить его успех на рынке. Вся работа технических специалистов пойдёт прахом, если специалисты по маркетингу не смогут сделать так, чтобы программный продукт заметили на рынке. Сотрудничество и командный дух в работе этих подразделений имеет решающее значение. Чем теснее они сотрудничают, чем в большей степени они единомышленники и чувствуют себя единым целым, тем больше шансов, что коррективы будут вноситься быстро и эффективно, повышая тем самым шансы на успех.

Между прочим, с отправкой программы заказчику работа над проектом не заканчивается. Проект можно считать завершённым, лишь когда продукт принесёт запланированную прибыль и решит поставленные перед ним экономические задачи. В конечном счёте успех определяется не фактом сдачи проекта, но завоеванием рынка.

Группа технической поддержки

Специалисту по технической поддержке, наверное, приходится чаще всех контактировать с клиентами после выхода продукта. Он изо дня в день является представителем группы разработчиков перед клиентами. Этот специалист играет очень важную и ценную роль не только для клиентов, но и для разработчиков. В частности, разработчики рассчитывают на помощь группы технической поддержки в решении следующих задач:

• формулирование требований к функциям программы с целью облегчения её технической поддержки в дальнейшем;

• привлечение внимания разработчиков к важным проблемам с качеством и реализацией функций во время бета-тестирования продукта и после его выхода;

• предоставление статистики поступающих обращений за технической поддержкой (упорядоченной по типу и степени серьёзности проблемы, а также по числу обращений) и демонстрация необходимости исправлений или изменений программы;

• помощь в решении проблем с пользовательским интерфейсом путём проверки ранних версий (альфа-версий) продукта, при заминках с тестированием и при исправлении ошибок — здесь группа технической поддержки выступает в роли независимого эксперта.

Чтобы справиться с этими задачами, следует назначить ведущего специалиста по технической поддержке разрабатываемого продукта. У него должны быть тесные связи с менеджером проекта и ведущими специалистами. Он будет участвовать в создании продукта с начала и до конца.

Во время разработки программы ведущий специалист по технической поддержке имеет доступ к исходным файлам, внутренней документации, планам и графикам. После отправки ПО заказчику он занимается решением неотложных проблем совместно с другими ведущими специалистами и регулярно предоставляет отчёты об успехах продукта в отрасли, подкреплённые данными, характеризующими продукт как с положительной, так и с отрицательной стороны.

Нельзя недооценивать важность группы технической поддержки. Она входит в состав технических подразделений и должна полностью интегрироваться в проводимую ими работу. Хорошая техническая поддержка позволяет преодолеть все недостатки продукта, обнаруженные после его выхода, и существенно уменьшить недовольство клиентов продуктом (если оно будет иметь место).

Администратор программы бета-тестирования

Отвечает за планирование, управление и исполнение программы бета-тестирования. Хорошо проведённая программа бета-тестирования способствует успеху продукта, обеспечивая поступление отзывов о нём из реального мира. Важность бета-тестирования столь велика, что я посвятил этому вопросу целую главу. Но пока просто рассмотрим основные обязанности администратора программы бета-тестирования:

• поиск, проверка квалификации и привлечение кандидатов в бета-тестеры;

• распространение инструкций и ПО среди бета-тестеров;

• рассылка кандидатам в бета-тестеры анкет и других необходимых материалов;

• опубликование результатов бета-тестирования внутри группы;

• постепенное усовершенствование процесса бета-тестирования.

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

• Слишком расплывчатый или, наоборот, чересчур жёстко определённый круг обязанностей

Даже самым талантливым людям нужно объяснять их роли и обязанности. Они должны представлять, что от них требуется, чтобы знать, чему посвятить своё время. Формулировки обязанностей должны быть подробными и ориентированными на решение тех или иных задач. Но не переусердствуйте с этим, иначе эти формулировки станут слишком формальными и жёсткими. Вряд ли нужно, чтобы участники группы лишь цитировали описание их задания, когда пора уже выпускать продукт. Главное — избежать основных просчётов при исполнении проекта, а не пытаться управлять всем и всеми, даже в мелочах (см. описание обязанностей специалистов, приведённое в этой главе);

• Дисбаланс подразделений

Одно лишь наличие модели, которая поощряет равновесие навыков и опыта специалистов различных подразделений не означает, что обеспечение проекта кадрами осуществлялось как надо. Постоянно следите за балансом ресурсов функциональных подразделений проекта. Например, хватает ли в группе тестировщиков для реализации проекта? Бессмысленно держать 4-5 тестировщиков на 100 разработчиков, даже если первые обладают необходимыми навыками. Отношение числа разработчиков к числу тестировщиков критично для проекта и должно быть сбалансировано. Значение этого отношения зависит от потребностей проекта, но большинство организаций стремится поддерживать его между 2:1 и 4:1. Даже если не соблюдать такое отношение, тестирование всё равно придётся проводить, только в этом случае оно займёт больше времени.

• Недостаток масштабируемости

Один из потенциальных недостатков модели организации проекта, описанной в этой главе, — слабая масштабируемость. При расширении числа участников проекта, скажем с 20 до 100, единственного менеджера уже будет недостаточно для работы проекта. К счастью, у этой проблемы есть два решения.

Первое — традиционное — подразумевает наращивание числа ведущих специалистов: разработчиков, тестировщиков, технических писателей и т.д. В отношении вверенных им групп они берут на себя значительную часть обязанностей, выполняемых менеджером проекта. Обычно это решение даёт неплохие результаты, особенно если проект остаётся однородным, т.е. все 100 человек работают над созданием одного и того же продукта. При наличии квалифицированных менеджеров эта модель может давать хорошие результаты, даже если организация становится очень большой.

Второй подход — создать несколько групп с вышеописанной структурой организации, небольшие или средние по размеру. Это особенно хорошо работает, когда в рамках проекта ведётся разработка независимых программ, например, двух продуктов, редакций или независимых компонентов одного продукта. Для работы над каждой из независимых задач можно выделить небольшое число людей и обойтись даже меньшей, чем показанная здесь, моделью. Сформировав несколько небольших групп, можно решить проблему с масштабируемостью, но при этом возникают другие проблемы, присущие этой модели. Так, организацию из 100 человек можно разбить на независимые группы по 6-7 человек, но они не смогут в полной мере задействовать в совместной работе инструменты, стандарты, выделенные средства, наработанные процедуры и информацию.

Один из наилучших способов справиться с этой задачей — назначить для каждого функционального подразделения (программистов, тестировщиков, технологов, разработчиков документации, инженерных психологов) квалифицированного эксперта, который возглавит процесс диагностики и разрешения общих проблем. К их числу можно отнести выбор общих инструментов для тестирования, создание общей испытательной лаборатории, определение стандартов документации, практичности ПО и многие другие вопросы. Эксперт в данной области работает со всеми группами. участвующими в решении общих проблем и реализует политику, повышающую производительность труда каждой группы. В некотором роде такая организация

напоминает концепцию средневековых гильдий. Например, если все банкиры какого-либо города хотели сделать местную торговлю более эффективной, они могли сформировать гильдию банкиров, чтобы совместно обсуждать способы улучшения и активизации банковской деятельности. Аналогичный подход годится для организации тестирования, создания документации, технологии разработки ПО и т.д. При наличии ведущих специалистов с солидным опытом работы в той или иной области и сильным руководством такую модель организации вполне можно задействовать в компании.

Ранжирование сотрудников и корпоративная культура

Ранжирование позволяет оценить эффективность отдельного сотрудника по размеру вклада и его значению для организации, не отрицая важности работы и личного вклада других участников группы. В основе оценки значения индивидуума — его вклад, а не выполняемые им обязанности или положение в иерархии компании.

Часть этой главы посвящена такому понятию, как корпоративная культура компаний, занятых в разработке ПО. Производительность труда команды и способность своевременно выпустить ПО зависит от корпоративной культуры труда не меньше, чем от любого другого фактора.

Ранжирование

Само по себе понятие рейтинга не ново. Одни из примеров — спортивные команды, возводящие самых результативных спортсменов в ранг «привилегированных». Их значение для команды настолько велико, что с ними заключаются контракты на особых условиях. В некоторых организациях, занятых в создании ПО, рейтинг воплощён в системе должностных титулов. Например, используют такие приставки к названию должности, как «специалист первого (или второго) ранга», «старший» или «главный» специалист. Во всех примерах ранг служит для решения важных кадровых вопросов.

Правила ранжирования

Правила ранжирования должны быть просты, не стоит чрезмерно усложнять их, чтобы не посвящать расчёту ранга большую часть своего времени. Как говорит мой опыт, проще всего вести ранжирование, приписывая сотрудников к одному из кругов: внутреннему, среднему или внешнему.

Внутренний круг

Его составляют сотрудники, наиболее важные для компании. Любой хороший руководитель или ведущий специалист должен знать, кто из участников команды вносит наибольший вклад и на кого можно всегда положиться. Эти люди — движущая сила проекта (а часто и всего бизнеса). Их участие имеет стратегическое значение для успеха работы команды, поэтому их нужно соответствующим образом выделять и поощрять.

Как правило, внутренний круг составляют самые старшие по должности и наиболее одарённые участники коллектива, обладающие наибольшим доверием. На рабочих собраниях они пользуются большим авторитетом при выборе стратегии, определении направленности продукта и решении других важных для компании вопросов. Возглавляя функциональные подразделения, они олицетворяют собой мастерство и опыт и могут «сделать игру» в самые сложные моменты. Эти люди в полной мере ощущают ответственность за создание продукта и делают всё возможное для его успеха. Они не раз выручали группу в прошлом и не раз сделают это в будущем.

Средний круг

Включает перспективных сотрудников. Эти люди могут быть не столь одарёнными, как члены внутреннего круга, и не иметь их навыков. Однако они очень важны для успеха проекта. Как правило, недостаток опыта по сравнению с людьми внутреннего круга компенсируется у них неуёмным энтузиазмом, заинтересованностью, большим потенциалом роста и амбициями.

Здесь также можно встретить людей, которые работают неплохо, но обычно не демонстрируют исключительных качеств. Они стабильны, надёжны и последовательны, но их никак нельзя назвать выдающимися. Создание текущей версии продукта во многом зависит от их способностей, однако их нельзя считать «ключевыми игроками».

Внешний круг

Внешний круг по большей части состоит из людей, новых для организации, которые ещё не оправдали ожиданий в полной мере. Какими бы многообещающими ни казались новые работники, пока они всё равно остаются неизвестными и неиспытанными. Нужно дать им время, чтобы влиться в работу организации и доказать свои способности делом.

Людей внешнего круга можно без особого труда заменить. Их внезапный уход из организации не будет иметь стратегических последствий для её успеха.

Для чего нужно ранжирование?

С помощью ранжирования легче распределить ограниченные ресурсы и решить, перед кем открыть новые возможности. Кроме того, оно помогает позаботиться о людях, внёсших наибольший вклад в успех организации, и избежать увольнения сотрудников, играющих ключевые роли в создании ПО. Обсудим некоторые области применения ранжирования.

• Поощрение заслуженных сотрудников

Ранжирование позволяет отметить заслуженных сотрудников. Если специалист-«суперзвезда» пять лет трудился на благо группы, то факт принадлежности этого человека к внутреннему кругу и особого отношения к нему важен не только для него, но и для других членов группы. Он свидетельствует о том, что организация видит личный вклад сотрудника, благодарна за него и вклад других людей со временем также заслужит признание.

• Распределение привилегий и ограниченных ресурсов

Когда в компании появляются новые привилегии и ресурсы, которые нельзя разделить поровну, возникает вопрос: кому отдать предпочтение? Хорошая мысль — определить достойных по их рангу. Неважно, что это будет: захватывающие исследования новой технологии, посещение выставки, собственный кабинет или встреча с клиентом на Гавайях, — ранжирование даёт упорядоченный список авторов наибольшего вклада в успехи компании, которые будут первыми кандидатами на получение привилегий.

Значит ли это, что все блага будут доставаться лишь людям внутреннего круга? Вовсе нет. Если все работники внутреннего круга уже обеспечены, а некоторая привилегия имеет особое значение для другого работника, наверное, разумно было бы отдать ему эту привилегию. Важно, чтобы все привилегии не доставались лишь одному кругу — вместе с членами внутреннего круга что-то должны получать и люди среднего и внешнего кругов. И всё же основной

принцип остаётся в силе: достойны заботы лишь те, кто заботится об успехе продукта.

• Планирование денежных компенсаций

Рейтинг также помогает спланировать повышения зарплаты, денежные и другие премии. Несомненно, наибольшее внимание при этом надо уделять людям внутреннего круга, чтобы выразить им благодарность за большой вклад в дело компании. Возможно, их следует внести в список на получение премий или подарков, а может быть, и в оба списка. Что бы вы ни выбрали, важно компенсировать затраченные усилия, чтобы люди видели признание их достижений.

Во вторую очередь следует позаботиться о членах среднего круга и наконец — о людях внешнего. Помните: слова «во вторую, и „в последнюю“ очередь вовсе не означают, что размер компенсации должен быть равным или меньше рыночного уровня. Вы не пытаетесь наказать людей внешнего круга, просто размер их вознаграждения не так велик, как у других. Сам факт, что повышение качества работы открывает доступ к более высокой компенсации, должен быть стимулирующим.

• Планирование кадровой политики

Один мудрец как-то сказал мне, что текучесть кадров подобна ветру: не беда, если он сорвёт большую часть листьев (т.е. если уйдёт много людей внешнего круга): пройдёт немного времени, и они вырастут снова. Намного хуже, если ветер повредит ветви (средний круг) или ствол (внутренний круг) — их трудно восстановить и они долго растут. Мораль проста: если компанию покидают люди среднего и внутреннего кругов — это признак серьёзной проблемы, которую нужно решить. Ранжирование помогает разобраться, где появилась текучка и какой ущерб она наносит группе.

Некорректное использование ранжирования

Некорректное ранжирование может привести к расколу и конфронтации в коллективе. Старайтесь избежать в этом вопросе лицемерия, ханжества и высокомерия. Когда люди слишком важничают и перестают понимать необходимость личного вклада для всеобщего успеха, не замедлят появиться самые разные проблемы. Помните: выделяя и вознаграждая чей-либо вклад, важно не вызвать зависти и вражды остальных участников коллектива.

Привилегии и ответственность

Большие привилегии означают и более высокую ответственность. В случае ранжирования как никогда верна старая поговорка: «положение обязывает». Не допускайте, чтобы привилегии, которыми наделены сотрудники в соответствии с их рангом, были больше возложенной на них ответственности. Если что-то одно начинает перевешивать, жди проблем, малых и не очень.

Из собственного опыта

В NuMega мы в шутку называли наших инженеров «богами», «полубогами» и «учениками богов». Хотя все работники компании были по крайней мере

хорошими, а большинство — отличными специалистами, не все вносили одинаковый вклад в дело компании.

Чем больше вклад сотрудника, тем больше у него было возможностей и свободы. Но взамен от них ожидали новых идей и способности возглавить остальных, при необходимости направить их усилия и помочь советом. Поскольку большинство наших «суперзвезд» были не только талантливы, но и скромны, признание их вклада редко вызывало у других отрицательные эмоции.

Когда люди меняются

Что происходит, когда член внутреннего круга начинает вести себя, как член внешнего? Или наоборот, когда новичок поражает всех качеством своей работы?

В первом случае лучше всего обсудить возникшие проблемы с этим человеком, представив ему конкретные факты в подтверждение снижения эффективности его труда. Часто такое обсуждение позволяет выявить источник проблемы: может, это упадок сил или какие-то личные проблемы? Вооружившись этим знанием, можно попробовать сразу же решить эту проблему с большими шансами на успех.

Во втором случае похоже, что вы наткнулись на «суперзвезду». Во-первых, важно убедиться, что продемонстрированные незаурядные результаты стабильны. Не стоит спешить с выводами, обычно лучше подождать конца цикла разработки. «Суперзвёзды» не вспыхивают на миг, но продолжают «светить» и постоянно проявляют себя. Если новый сотрудник продолжает демонстрировать выдающиеся результаты, следует с радостью принять его во внутренний круг.

К чему стремиться

Естественно, конечная задача — перевести всех работников во внутренний круг. Самый лучший исход, которого можно желать — это формирование стабильной команды, состоящей сплошь из людей внутреннего круга. Не следует искусственно ограничивать число людей внутреннего круга. Обычно во внутреннем круге меньше сотрудников, чем в среднем, а в среднем — меньше, чем во внешнем. И всё же не следует насильственно поддерживать такое распределение — оно должно отражать реальный вклад участников группы.

Корпоративная культура

Понятие «культура» в приложении к созданию программ включает ряд определённых черт и понятий, влияющих на процесс разработки ПО. Культура формируется прежде всего принципами и действиями руководящего звена организации. Со временем культура растёт или приходит в упадок в зависимости от прошлых успехов и неудач группы, реакции на возникающие проблемы и способности решать поставленные задачи. Эти факторы в конечном счёте определяют самооценку группы. Она может быть любой: участники группы могут считать себя специалистами высшего пилотажа, создающими замечательные продукты, или страдальцами, еле справляющимися со своей работой.

Почему культура так важна?

Как и у индивидуума, так и у команды должно быть представление о предназначении, важности, уровне мастерства и способности решить поставленную перед ней задачу. Если самооценка команды высока, её мотивация и работоспособность достаточны, чтобы с высокой производительностью труда создавать высококачественные продукты. И наоборот, если самооценка низка, команда будет работать ниже своих способностей, даже если она состоит из талантливых людей! В определённом смысле культура может как повышать, так и снижать общую работоспособность коллектива.

Высокая культура разработки ПО, в частности, имеет огромное значение при освоении новых возможностей или для выхода из затруднительной ситуации. Если в компании развита культура создания качественных продуктов, весьма вероятно, что внутреннее стремление и приверженность к этой практике не иссякнут и в будущем.

С другой стороны, низкая культура лишь осложняет и без того трудную ситуацию. Допустим, команда, в которой никогда не воспитывалась культура быстрой реакции на внешние события, оказалась в ситуации, требующей немедленных действий. В этом случае успех попытки быстро отреагировать на событие представляется весьма сомнительным; такой группе, возможно, даже не стоит и пытаться сделать это.

Как воспитать корпоративную культуру?

Лучший способ создать и воспитать корпоративную культуру — начать с приверженности определённым принципам и культивирования некоторых манер поведения, после чего группа какое-то время должна работать по новым правилам. Действия группы в соответствии с этими принципами и будут факторами, формирующими её культуру.

Из собственного опыта

Воспитывая корпоративную культуру NuMega, мы специально предпринимаем ряд конкретных действий.

• Чётко определяем, кто мы и какие цели преследуем

Нам совершенно ясно, кто мы и что должны делать. Звучит довольно просто, да? Но на деле это не так просто, как кажется.

Мы стремимся создавать лучшие в мире средства разработки, в частности инструменты, которые помогают разработчикам отлаживать программы и устранять в них неполадки. Мы также знаем, что любые методы достижения этой цели автоматически попадают в наше поле зрения. Мы должны разобраться в них, перенять и применять их лучше, чем кто-либо другой.

• Культивируем элитарность

Строгий отбор при приёме на работу позволяет многим считать привилегией принадлежность к числу технических специалистов нашей компании. Культивирование окружения, построенного на привилегиях, позволяет поддерживать высокий боевой дух и снижать текучесть кадров. Элитарность

позволяет воспитать такие черты культуры, как приверженность общему делу, стремление делать заметные вещи и участвовать в решении нестандартных задач.

• Отмечаем успехи и помним свою историю

Со временем в компании сформировалась история выпуска замечательных продуктов, успешных как в техническом, так и в экономическом отношении. «Отцы-основатели» постоянно приводят эти успехи в качестве примеров на собраниях и неформальных встречах. Все следят за тем, чтобы каждый новичок знал историю и путь развития компании. Это стимулирует формирование культуры стремления к успеху, создания замечательных продуктов и заставляет быть экспертом в выбранной нами области.

Эти принципы имеют большое значение, ибо история компании влияет на формирование культуры. Обязательно отмечайте успехи и передавайте историю компании новым работникам.

• Поддерживаем дух соперничества

Ставя цель перед коллективом, мы чётко определяем правила соревнования и относимся к поставленным целям, как к врагам, после чего, «сомкнув ряды», атакуем их. Цель соревнования очень чётко обозначена и пронизывает предметы наших многочисленных обсуждений и размышлений. Порой обсуждения проходят так оживлённо, что мысли переливаются через край, принимая причудливые формы; чего стоят те трюки, которые творят разработчики. Наглядевшись на это, я понял: эта команда сделает всё возможное и невозможное, чтобы добиться успеха, и нам удалось сформировать замечательную культуру соперничества.

• У нас свои способы отдохнуть и весело провести время

У работников технических компаний есть два удовольствия: возможность от души повеселиться и разные способы отдохнуть от повседневных забот. У нас куча всяких вещиц: футболок, рекламных материалов с выставок, бесплатных обедов, кроме того, «NuMega specials» (пицца с поджаренным луком, чесноком и жгучим «чили»), самодельные площадки для гольфа и велосипедные дорожки прямо в помещениях компании. Всё это формирует атмосферу исключительности и веселья. Мы знаем, что это очень нетипично, но это нам и нравится!

• Не прячем своих разработчиков от клиентов

В NuMega искренне убеждены, что разработчиков не следует прятать от клиентов. Проще всего попросить их присутствовать на стенде компании на какой-нибудь выставке. Там они смогут не только отдохнуть от работы, но и попадут на передний край общения с клиентами, смогут ответить на их вопросы, получить новые идеи и услышать о созданных ими продуктах как хорошее, так и плохое.

Такие выставки поднимают боевой дух разработчиков до небес, поскольку каждый из них получает сведения о потребностях клиентов из первых рук. Таким образом удаётся наладить живую связь между теми, кто создаёт программы, и теми, кто ими пользуется.

***

Немного конкретизируем изложенное выше. Первый этап формирования культуры — определение приоритетных ценностей. Какие ценности заложены в

культуре вашей компании в данный момент? Какие ценности должны быть воплощены в будущем? Подумайте, можно ли сказать, что ваша команда:

• ценит выполненную вовремя работу?
• ценит техническое превосходство?
• ценит высокое качество?
• ценит даже минимальный вклад?
• поощряет риск?
• поощряет исключительную производительность труда?
• проявляет благородство?
• небезразлична к социальным проблемам?
• считает, что каждый должен принимать участие в тестировании продукта?
• оперативно реагирует на внешние угрозы?
• считает, что тестирование практичности программы имеет решающее значение?
• считает сверхурочную работу обычным делом при отставании от графика?

Следующий этап — выбор действий, формирующих корпоративную культуру. Какие препятствия вы сможете преодолеть при этом, какие решения сможете принять, какие проблемы затронуть и на какие конфронтации пойти ради воспитания необходимой культуры? Подумайте об этом и приготовьтесь последовательно воплощать принятое решение. Помните: в конечном счёте культура все равно возникает, пытаетесь вы направлять её формирование или нет.

Корпоративная культура и технологические приёмы

Другой аспект культуры состоит в использовании и освоении внутренних технологических приёмов. Характерной чертой некоторых культур является наличие множества технологических приёмов разработки, в то время как у других их почти нет. Молодые компании часто неохотно осваивают новые технологические приёмы, тогда как в более крупных организациях без них не обходится ни одно задание. По мере роста группы следует подумать о том, как вы собираетесь осваивать новые технологические приёмы. Какие типы приёмов необходимы, а без каких можно обойтись? Как не впасть в крайность, пустив всю работу на самотёк или изнурив себя разными правилами и положениями?

Вот некоторые правила, которых полезно придерживаться при росте организации:

• Взвешивайте затраты на внедрение технологических приёмов и пользу от них

Необходимо сразу распознавать ситуации, когда добавление нового приёма не принесёт пользы. Польза от каждого дополнительного этапа или процесса должна окупать затраты на его внедрение. Порой это трудно выяснить сразу, и в сомнительных случаях лучше отказаться от внедрения новшества.

Если вы чувствуете, что новый процесс необходим, важно знать, как его внедрить. Новые технологические приёмы приносят наилучшие результаты, когда они чётко определены, а их внедрение сулит очевидную выгоду (обеспечивая реальные и измеримые результаты) и не встречает сопротивления коллектива. Если новый приём соответствует этим критериям, велик шанс, что его можно будет с успехом использовать.

С другой стороны, если приём описан туманно, не имеет очевидной ценности или команда не принимает его, следует ещё раз спросить: а нужен ли он? Я не имел в виду, что следует вовсе отказаться от его внедрения, просто нужно быть уверенным, что выгода от внедрения перевесит затраты. Также неплохо было бы проработать проблемы, которые могут встретиться на пути внедрения нового приёма. Не следует спешить с добавлением новых приёмов, но как только исчезли последние сомнения в их необходимости, следует без колебаний приступать к внедрению.

• Прежде, чем внедрять новые приёмы, нужно извлечь максимум из имеющихся

Один из самых обескураживающих аспектов разработки ПО — внедрение новых приёмов, в то время как команда не полностью использует все возможности имеющихся. В вашей команде так быть не должно. За правильное использование имеющихся приёмов отвечают менеджеры и ведущие специалисты. Можно потерять доверие участников команды, санкционируя добавление новых технологических приёмов и игнорируя те, что уже есть. Если обнаружится, что ни один из имеющихся документированных приёмов не используется полностью, созовите собрание группы и решите, представляют ли они ценность. Да — оставьте их и используйте «на все сто». Нет — избавьтесь от лишних технологических приёмов. Что толку держать приёмы, развесив их по стенам, как картины? Они только портят интерьер.

Суть в следующем: нужно внедрить минимальный набор технологических приёмов, необходимый для выполнения работы и следить за тем, чтобы группа освоила их в совершенстве. Не следует осваивать новые приёмы, если вы сами не уверены, что они будут задействованы в полной мере. Группа должна знать: уж если вы решили что-то сделать, то сделаете это непременно.

• Взвешивайте потребности отдельного специалиста и всей команды

Обычно новые приёмы осваиваются, чтобы удовлетворить потребности всей команды. Однако порой новые приёмы вводятся, лишь чтобы облегчить жизнь нескольким людям или даже одному человеку. В этих случаях надо тщательно проанализировать все затраты и выгоды. Если для внедрения этого приёма придётся просить группу сделать что-то сверх запланированного с существенными затратами, нужно быть уверенным, что польза для немногих участников группы стоит затрат, на которые придётся пойти остальным. Если они несоизмеримы, от нового приёма следует отказаться, в противном случае надо разъяснить причины всем, кого коснётся внедрение нового приёма и кому придётся вкладывать в него своё время и силы, чтобы обеспечить его стопроцентное использование.

Из собственного опыта

Как-то раз в NuMega было предложено при регистрации каждой ошибки сопровождать её сведениями о программно-аппаратной конфигурации компьютера, на котором она была обнаружена. Предложение должно быть полезным для тех, кто смог бы легче находить определённые типы ошибок с его

помощью и прекрасно подходит для разнородных сред или в случае команд, где разработчики рассредоточены. Однако это стало бы настоящим препятствием для остальных участников группы, которым приходится вводить информацию вручную и без ошибок. Но нашу среду нельзя назвать ни разнородной, ни географически рассредоточенной.

Представляет ли новый приём ценность для компании? Иногда — да, когда дополнительная точная информация позволит легче устранять ошибки, но число таких ошибок не превышает 5-10 в месяц. Однако затраты на ввод этой информации, который должны были выполнять 15 человек на протяжении всего внутреннего цикла разработки (особенно когда приходилось регистрировать до 10 ошибок в день), не стоили потенциальной пользы. Кроме того, большинство ошибок воспроизводилось на любой машине, в противном случае было нетрудно связаться с офисом, откуда поступило сообщение об ошибке, и выяснить необходимую информацию о конфигурации.

Отвергнув это предложение и множество ему подобных, нам удалось поддерживать накладные расходы низкими и в полной мере задействовать имеющийся арсенал технологических приёмов.

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Проблемы с ранжированием

• Не держите ранжирование в секрете

Не скрывайте факт, что в компании ведётся ранжирование сотрудников, хотя вывешивать транспарант с рангами на виду у всех тоже не стоит. Людям важно знать, что их личный вклад востребован, будет оценен по достоинству и его учтут при распределении различных привилегий и компенсаций.

• Не затрудняйте переход из одного круга в другой

Не стройте искусственных преград между внешним, средним и внутренним кругами. Единственным способом перехода в высшие круги должно быть повышение личного вклада, а единственной причиной перевода в низшие — недостаточный вклад. Кроме того, обязательно нужно отмечать стабильные результаты переводом из внешнего круга в средний, а хроническое снижение качества работы — переводом из внутреннего круга во внешний.

Как сказано выше, при переходе работника из одного круга в другой непременно нужно обсудить это с ним. Позитивные переходы должны быть отмечены и поощрены. Негативные переходы тоже следует отметить и обсудить с глазу на глаз.

• Не заводите фаворитов

Не допускайте использования рангов для определения фаворитов и не путайте его со средством вознаграждения любимчиков. Ранг должен быть основан исключительно на личном вкладе работника. Если ваш друг из группы не вносит вклад наравне с остальными, он не должен участвовать в получении благ наравне

с остальными, и точка. Если вы не в состоянии пойти на такое решение, следует вовсе отказаться от ранжирования.

• Играйте честно

Некоторые считают ранжирование сплошным обманом. Причина может быть в том, что ранее этим людям уже приходилось сталкиваться с некорректным использованием ранжирования (см. обсуждение фаворитизма выше). Но с другой стороны, ранжирование никогда не бывает честным, если под честностью понимать уравниловку. Ранжирование призвано не уравнять всех, но помочь при распределении ресурсов на основе размера личного вклада, отметить выдающиеся достижения и стимулировать дальнейшие успехи. В мире, где ресурсы ограничены, нужно найти способ распределения ресурсов, и, вероятно, нет ничего лучше, чем распределение на основе вклада в создание продукта или работу коллектива.

Проблемы с культурой

• Культуру можно изменить

Одна из самых серьёзных проблем с культурой — в ощущении её неизменности. Корпоративная культура больших коллективов и компаний кажется особенно непоколебимой. Хотя изменить культуру может быть трудно, это вполне возможно, если руководство всерьёз на это решилось. Кроме того, в отдельных подразделениях часто складывается собственная микрокультура, созданная в рамках корпоративной культуры, изменить которую сравнительно легко.

• Если организация растёт

По мере роста организации очень важно прививать корпоративную культуру новым работникам. Особенно часто быстрый рост штата отмечается в начинающих компаниях, поэтому там следует особенно тщательно следить за сохранением основных корпоративных ценностей. Для этого следует не забывать о таких вопросах:

— кто мы?

— чем мы занимаемся?

— почему мы делаем своё дело лучше, чем кто-либо другой?

— чем мы отличаемся от других?

— что мы ценим?

Проследите, чтобы все работники компании знали ответы на эти вопросы и эти знания передавались новичкам. Будет ли это собрание с участием участника группы администраторов или просто обсуждение коллегами по команде — вопросы культуры непременно следует обсуждать в открытую и со всей серьёзностью, которой они заслуживают.

Инструментальные программы

В компании NuMega мы использовали лишь те инструментальные программные средства, которые были жизненно необходимы для нашей работы и вписывались в наш стиль работы. Из всех доступных инструментов мы больше всего применяли и полагались на систему управления исходным кодом и систему устранения проблем и неисправностей (поиска «жучков»). Возможность приспосабливать эти продукты к нашим нуждам позволила ускорить работу — вся команда использовала их почти каждый день.

Средства управления исходным кодом

Ваш исходный код — это второй наиболее важный актив проекта, после людей, конечно. Следовательно, во всех проектах, связанных с разработкой ПО, даже в тех, где задействован всего один человек, должна быть обеспечена целостность исходного кода. В течение цикла разработки вам потребуется проверять, обновлять, контролировать и пересматривать изменения в исходном коде. С ростом количества людей, работающих над проектом, и сложности проекта эти требования станут ещё более критичными. Мы рассмотрим основные возможности программного обеспечения по управлению исходным кодом и обсудим некоторые простые приёмы, позволяющие максимально увеличить его полезность.

Продукты по управлению исходным кодом хранят файлы с кодом, отслеживают их версии, управляют файлами, составляющими проект, и предоставляют следующие функции.

• Хранение файла и его прошлых, настоящих и будущих версий

Система управления исходным кодом будет обслуживать все порученные ей версии файлов. Она сможет выдать любую версию файла, размещённую в системе. Эта возможность необходима, если вы собираетесь строить приложение на основе предыдущих версий, и особенно важна при одновременном создании нескольких версий одной программы.

• Отслеживание истории изменений для каждого файла

При любых изменениях в файлах система управления исходным кодом внесёт нужные сведения в историю изменений: дату, время, пользователя и обычно небольшое примечание пользователя о природе изменения и его масштабе. Часто эти комментарии — единственное, чем вы располагаете. (Эта информация поможет новым разработчикам втянуться в проект.)

• Группирование файлов

С ростом сложности проекта возникает необходимость группировать файлы: по назначению (например, по подпроектам или подсистемам) или по функциям (например, тестовые задания, спецификации и документация).

• Маркировка файлов, связанных с конкретным выпуском программного продукта

Система управления исходным кодом позволит пользователям пометить определённые версии файлов всего проекта/подпроекта. Это позволяет чётко

маркировать или идентифицировать файлы, составляющие определённый выпуск ПО.

• Блокировка и разблокировка файлов

В процессе разработки доступ к рабочему набору файлов требуется более чем одному человеку. Если какой-то файл не используется кем-то ещё, система управления исходным кодом заблокирует файл и выдаст его пользователю для работы. Это действие предотвратит изменение и возможную порчу файла другими пользователями. Когда тот, кому выдан файл, завершит свою работу и возвратит файл в систему, файл будет разблокирован, и появится возможность доступа и выдачи этого файла другим пользователям. Иногда двум разработчикам необходимо одновременно редактировать один файл. Для таких случаев имеется возможность обойти блокировку файла, но тогда координировать все изменения придётся вам. Наиболее изощрённые продукты по управлению исходным кодом обеспечивают множественную выдачу файлов, автоматически совмещая изменения, сделанные в двух файлах. Однако это может быть опасно, так что большинство разработчиков выполняют операции по слиянию вручную (контролируют этот процесс).

Что туда входит

Удобное расположение всех файлов и информации, связанной с проектом — это страшная (но излечимая) головная боль при разработке ПО. В NuMega не было времени создавать обширную инфраструктуру или сложные процессы для поддержки грамотного документооборота. Поэтому мы решили просто поместить все файлы и документы проекта в систему управления исходным кодом. Это были:

• исходные файлы;
• файлы заголовков;
• файлы библиотек;
• сценарии компоновки;
• результаты компиляции;
• результаты компоновки;
• инструменты и файлы для установки программ;
• инструменты и файлы для тестирования;
• спецификации проекта;
• планы проекта (ПО, документации и тестирования);
• пользовательская документация;
• тестовые задания и сценарии;
• тестовые модули разработчиков.

Из собственного опыта

Мы думали, что поместили все нужные файлы в систему управления, однако забыли о тестовых модулях, созданных разработчиками. Тестовые модули — это короткие программы для проверки функциональности продукта. Разработчики запускают их в процессе разработки, чтобы убедиться, что программа все ещё работает. Тестовые модули особенно полезны при добавлении новых возможностей и исправлении ошибок. Разработчики со стажем имеют в

своём арсенале десятки маленьких программ, осуществляющих проверку различных условий и вариантов, эти программки никогда не входят в официальные планы тестирования. Если бы эти тесты хранились централизованно, например, в системе управления исходным кодом, мы бы могли запускать их автоматически как часть нашего набора тестов контроля качества. Кроме того, эти тесты могли бы снизить риск добавления новых ошибок неопытными программистами, работающими над хрупкими или сложными частями кода.

Заметим, что в систему управления исходным кодом мы поместили всё, что можно было представить, а не только исходный код. Если что-то использовалось в проекте или относилось к проекту, мы помещали это в систему.

Зачем это нужно

При помещении всех файлов проекта в систему управления исходным кодом вы получаете большие преимущества. Во-первых, неважно, что это за файл и когда вы подключились к работе над проектом, — вы почти наверняка найдёте нужный файл в системе управления исходным кодом, а это весьма ценно. Разработчик сможет найти планы тестирования, технический писатель — функциональные спецификации, а новый сотрудник — нужную информацию без предварительного знакомства с историей создания проекта. Во-вторых, храня все файлы проекта в системе управления исходным кодом, вы сможете использовать функции маркировки и блокирования файлов, управления версиями и ведения истории.

Например, в NuMega была возможность отслеживать изменения в планах, тестовых сценариях и документации для всего проекта. К тому же мы могли помечать целые наборы тестовых сценариев и пользовательской документации для каждого внутреннего этапа и для каждого основного или неосновного выпуска. Мы могли с точностью до символа воссоздать все файлы (не только файлы исходного кода) любого выпуска в истории проекта.

Особо отмечу включение в систему NuMega средств сборки: компиляторов, компоновщиков, заголовков и т.д. Чёткое управление этими средствами критично для обслуживания сборочной среды проекта. Официальная сборочная среда была всегда доступна в системе управления исходным кодом. Все разработчики и машины, на которых собирался проект, должны были использовать один и тот же набор файлов. Без исключений.

Так как для сборки продукта мы не полагались на локальные файлы (не содержащиеся в системе управления версиями), мы задействовали идентичную сборочную среду во всём проекте. Этот простой подход сэкономил нам немало времени. До применения такой системы мы постоянно сталкивались с проблемами при сборке из-за несовместимости между компиляторами разработчиков или искали труднонаходимые ошибки во время выполнения, вызванные несоответствием библиотек или заголовков.

И последнее (но не менее ценное) преимущество — централизация файлов в системе управления исходным кодом обеспечивает простое резервное

копирование всего проекта. Одной командой мы могли создать резервную копню или просто скопировать весь проект на другой диск или другую машину.

Каковы их технологические возможности

Помимо функциональных возможностей, описанных ранее, команда в NuMega нуждалась в поддержке пяти жизненно необходимых технологических возможностей. Хотя они специфичны для наших продуктов и компании, многие из них стандартны для большинства проектов в отрасли. Это:

• управление разработкой нескольких редакций продукта;

• управление разработкой нескольких версий одной редакции;

• применение общих компонентов, как в рамках одного, так и нескольких проектов компании;

• компоновка продукта на основе самого свежего набора файлов с исходным кодом (или на основе другого набора исходных файлов);

• поддержка локальных сборок для отдельных разработчиков.

Как ими управлять

Одна из главных задач в управлении проектом по разработке ПО — это контроль сложности проекта. Труднее всего справляться с исходным кодом и управлением конфигурацией. Хотя наше решение было не идеальным, оно все равно работало и помогло нам без особых проблем укладываться в сроки.

Мы использовали систему управления исходным кодом Visual Source Safe (VSS) компании Microsoft. Она предоставляет нужные нам основные функции, и у неё отличная цена — как раз для начинающего бизнеса. Хотя обсуждение VSS выходит за рамки этой книги, я расскажу о том, как мы приспособили этот продукт под наши нужды.

Основы структуры

Мы структурировали наши проекты по двум простым элементам: частям и продуктам.

Часть — это компонент, используемый для компоновки программного продукта. Частями могут владеть разработчики, не являющиеся членами команды, работающей над проектом. Они могут обновлять свои части по собственному (однако согласуемому) графику, отличному от графика всего проекта. Продуктом являлся конечный пакет, продаваемый пользователям. Он складывался из уникальных для этого продукта частей и файлов. Храня в одном месте части и продукты, мы могли собирать различные редакции наших программ и одновременно поддерживать несколько параллельных направлений в разработке. Например, возможность выпускать исправления или пакеты обновлений, продолжая направлять все силы на разработку нового кода, была необходима и для поддержки, и для получения прибыли от следующих продуктов.

Структура и использование хранилища исходного кода

Все файлы, используемые при разработке наших продуктов, были рассортированы по трём папкам:

• Product Name — для файлов, относящихся к продукту;

• Environment — для файлов среды разработки;
• Imports — для сторонних файлов.

Папка Product Name содержала создаваемые нами файлы, необходимые для сборки, тестирования и написания документации к продукту (табл. 5-1). В ней были подкаталоги Branch для каждого варианта, над которым мы работали. В подкаталоге Parts хранились стандартные и совместно используемые компоненты, включаемые в продукт. И, наконец, для каждой редакции продукта имелся подкаталог Product. В каждой папке Product содержались необходимые для продукта части. Чтобы эта структура работала, нужно строго соблюдать соглашения об именах и не нарушать структуру. Координация изменений в частях и продуктах также была критичной.

Табл. 5-1. Примерная структура папки «Product Name».

$/Product_Name/ — Файлы, относящиеся к продукту

$/Product_Name/Branch/ — Различные направления в разработке

$/Product_Name/Parts/ — Совместно используемые файлы, входящие в продукт

$/Product_Name/Parts/Src/ — Исходный код для Parts (при необходимости совместно используется с/Products/Src)

$/Product_Name/Parts/Doc/ — Исходные файлы документации

$/Product_Name/Parts/Help/ — Исходные файлы справочной системы

$/Product_Name/Parts/Install/ — Исходный код программы установки

$/Product_Name/Parts/Patch/ — Исходный код вставок

$/Product_Name/Parts/Setup/ — Исходный код установщика

$/Product_Name/Parts/Samples/ — Исходный код с примерами

$/Product_Name/Parts/Tests/ — Исходный код тестов, тестовые задания и т.д.

$/Product_Name/Product/ — Редакции продукта Product Name (по одной папке на каждую редакцию)

$/Product_Name/Product/Output/ — Область для программ, созданных в других проектах

$/Product_Name/Product/Src/ — Исходный код продукта (при необходимости совместно используется с /Parts/Src)

$/Product_Name/Product/Doc/ — Файлы документации к продукту (совместно используется с /Parts/Doc)

$/Product_Name/Product/Help/ — Файлы справочной системы продукта (совместно используется с /Parts/Help)

$/Product_Name/Product/Imports/ — Импорт (совместно используется с /Imports)

$/Product_Name/Product/Install/ — Файлы для установки продукта (используется с /Parts/Install)

$/Product_Name/Product/Samples/ — Примеры для продукта (совместно используется с /Parts/Samples)

$/Product_Name/Product/Tests/ — Тестовые задания, тестовые сценарии (совместно используется с /Parts/Tests)

Из собственного опыта

Когда я пришёл в NuMega, для одного из продуктов было создано три каталога по именам разработчиков: Frank, Bill и Matt. Так как каждый работал над своим собственным кодом, они могли вносить изменения, не повреждая чужих файлов. Однако там было мало общего кода (одна большая структура данных для основных подсистем). Но это работало! Дальше нам нужно было увеличить команду разработчиков, и мы уже не могли обойтись без системы управления исходным кодом. Такая система позволила усложнить проект, удерживая его под контролем. Без неё я просто не могу представить ПО для разработчиков.

В папке Environment ($/Env) хранятся файлы, используемые командой разработчиков, но не являющиеся частью конечного продукта. Это все, начиная с инструментов и утилит и заканчивая стандартами создания кода и шаблонами для проекта. Папка Environment содержит файлы среды, описывающие среду с точки зрения разработчика, а также с точки зрения тестирования и документации. В NuMega мы хотели создать общую среду для команд разработчиков, и потому для этой цели мы создали специальный раздел в хранилище исходного кода. Вот примерный список подкаталогов, которые могут быть в этом разделе хранилища исходного кода (табл. 5-2):

Табл. 5-2. Примерная структура папки Environment.

$/Env/Dev/ ПО среды разработки и инструментальных средств

$/Env/Dev/Bin Исполняемые модули (подкаталог для каждого инструмента)

$/Env/Dev/Src Исходный код для этих инструментов (подкаталог для каждого)

$/Env/Dev/Doc Документация для этих инструментов (подкаталог для каждого)

$/Env/Dev/Etc Прочие файлы

$/Env/Test/ Инструментальные средства и файлы для тестирования

$/Env/Test/Bin Исполняемые модули

$/Env/Test/Src Исходный код и документация для этих инструментов

$/Env/Test/Doc Документация по среде тестирования

$/Env/Test/Etc Прочие файлы

$/Env/Documentation/ Документация по среде проекта

$/Env/Documentation/Templates Шаблоны проекта, шаблоны документации и справочники по стилям

$/Env/Documentation/Plans Планы и спецификации проекта, тестовых заданий и документации

$/Env/Documentation/Process Технологические документы проекта

$/Env/Etc Прочие файлы, не вошедшие в предыдущие категории

В папке Imports ($/Imports) хранились файлы или наборы инструментов из сторонних продуктов (табл. 5-3). Сами сторонние продукты в этой папке не содержались, там были только библиотеки и заголовки. В результате в разделе Imports проводилось совсем немного изменений. Однако так как эта область использовалась для хранения различных версий сторонних продуктов, было очень важно не вносить никаких изменений без чёткого осмысления, полного тестирования и учёта связей с элементами, на которые эти изменения могли бы подействовать.

Табл. 5-3. Примерная структура папки Imports.

$/Imports/RogueWave Библиотеки и заголовки RogueWave

$/Imports/ObjectSpace Библиотеки и заголовки ObjectSpace

$/Imports/Visual C Результаты компиляции, библиотеки и заголовки Visual C

$/Imports/Install Shield Библиотеки и заголовки Install Shield

$/Imports/Прочие Папки для каждого инструмента или библиотеки сторонних производителей

Компоновочная система

В NuMega мы писали сценарий сборки продукта на выделенной «компоновочной машине». Сценарий должен был выбирать нужные для сборки продукта файлы из системы управления исходным кодом. Эта информация включала как сами средства компоновки, так и исходные файлы, библиотеки, заголовки и другие необходимые компоненты. Для ведения процесса компиляции мы выбрали Nmake — популярное средство управления компоновкой. Nmake сначала собирает все части продукта, а затей компонует окончательные исполняемые файлы продукта.

Сценарий компоновки в качестве входных данных принимал метку сборки, позволяло нам создать определённые версии продукта. Так как мы маркировали и отбирали и средства сборки, и файлы продукта, то таким образом мы гарантировали надёжность сборочной среды. Сценарии компоновки также использовали стандартные переменные окружения и макросы, так что мы собирали все части и продукты посредством одного вызова. То, что наша компоновочная машина и разработчики использовали одни и те же сценарии компоновки, позволяло собирать файлы проекта просто и без ошибок.

Надёжная автоматическая система компоновки — необходимое условие успешной разработки. Затраты времени и сил на то, чтобы заставить эту систему работать, своего стоят. Этот и другие вопросы применения технологического ПО обсуждаются в главе 7.

Устранение проблем и неисправностей

Разработка ПО — это динамичный процесс с интенсивным обменом информацией между его участниками. При работе команды над проектом очень важно определить формальные методы поиска и устранения проблем, неисправностей и «жучков», которые постоянно появляются. Применение специальных продуктов для устранения проблем и неисправностей — один из лучших выходов. В оставшейся части главы объясняется, как эффективно использовать такие продукты.

ПО для устранения проблем и неисправностей позволяет справляться с бесконечными ошибками и проблемами, всплывающими на поверхность в процессе разработки. Эти программы позволяют членам команды протоколировать, обновлять, назначать, устанавливать приоритет, сортировать и пересматривать информацию, полученную в цикле разработки проекта. Они являются важной частью любого проекта независимо от его размера. Никто не в состоянии запомнить все ошибки и проблемы, которые следует разрешить. И если

они не протоколируются, не пересматриваются и для них не устанавливается приоритет, качественного продукта не создать.

Что туда входит

В NuMega использовали систему устранения проблем и неисправностей для хранения любых постоянных или временных данных о проекте, которые только можно было представить. Туда входили все программные ошибки (в том числе функциональные, затрагивающие производительность, процесс установки и параметры, а также все ошибки при сборке) и решения или предложения по улучшению проекта, его настоящих и будущих версий. Здесь действует простой, но очень важный принцип: все данные должны храниться в одном месте. Не следует держать их в хранилище, не обеспечивающем совместное использование, резервное копирование и простой доступ. (Поэтому сообщения электронной почты не подходят!)

Примечание

Я бы не советовал писать собственные программы по устранению проблем и неисправностей, так как хватает различных коммерческих программных продуктов по разумным ценам. Например, Compuware/NuMega TrackRecord, PVCS Tracker, Rational ClearQuest и др. Хотя вам потребуется самим определить, какая из них отвечает вашим потребностям, я настоятельно рекомендую принять решение в пользу покупки. Ведь вы не обязаны тратить время на создание собственных инструментов, ваше дело — поставлять ПО.

Рассмотрим пример. Разработчик замечает, что производительность в одной из частей приложения здорово снизилась, и сообщает об этом по электронной почте в конференцию разработчиков. А дальше? Кто заметит сообщение, а кто и нет. Даже если кто-то находит неполадку, её нужно запротоколировать и устранить, иначе о проблеме просто забудут. Послать сообщение по электронной почте — не плохо, но не зафиксировать наличие неисправности — беда. То же касается предложений по выпуску. Есть вероятность, что на предложение никто не откликнется, и если сообщение электронной почты было послано без протоколирования, то не будет и никакой истории работы над предложением.

Как это работает

Приведённый пример показывает, насколько важно автоматизировать элементарный обмен информацией между членами команды. Запомните: инструмент должен работать на вас, и никак иначе. Вам нужно управлять информацией просто и без лишних формальностей. В то же время вы должны следить за соблюдением дисциплины и не допускать небрежности. Для этих целей используется система устранения проблем и неисправностей. Сконфигурируйте её так, чтобы собирать следующие основные данные о проекте:

- текущее состояние проблемы: открытая или закрытая;
- дата возникновения, изменения и решения;
- текстовое описание проблемы;
- номер выпуска/сборки программы, в которой обнаружена проблема;
- имя человека, описавшего проблему:
- имя человека, работающего над проблемой в настоящее время;

• состояние проблемы: расследуется, требуется больше информации, ожидается внешнее событие, решена и т.д.;
• приоритет проблемы: низкий, средний, высокий;
• выпуск, в котором присутствует проблема;
• статут процедуры контроля качества;
• количество попыток неуспешного решения проблемы;
• список изменений к отчёту о проблеме или неисправности.

Посмотрим, как мы использовали ПО для устранения проблем и неисправностей в NuMega.

Из собственного опыта

Когда я начинал работать в NuMega, «жучки» и другие различные неисправности фиксировались на доске (если вообще фиксировались). Они оставались там до тех пор, пока не были исправлены, а затем их просто стирали. Когда доска заполнялась полностью, новые записи втискивали в уголок или, чтобы освободить место, стирали другие ошибки. Эта система работала для одного-двух человек, но истории работы над ошибками не было.

С ростом организации становилось очевидно, что нам требуется автоматизированное решение. Если бы мы в то время не перешли к нему, то не смогли бы успешно вырастить свою команду. Хотя поиск инструмента, способного решить наши проблемы, и не был сложен, способ его использования часто становился предметом спора. В конце концов мы выяснили, что для группы нашего размера куда важнее выполнять на «отлично» всего несколько функций, чем пытаться делать всё, что мы можем только представить.

Для всех ошибок — одно хранилище

У нас было простое правило: обо всех ошибках сообщать системе устранения неполадок. Если их там нет, значит, их не существует. Это здорово упростило управление мероприятиями по исправлению ошибок. Сплетни, кулуарные разговоры и сообщения электронной почты не годятся в качестве методов протоколирования ошибок. Скажем, если информация о неисправности приходит от службы технической поддержки, управления продуктом, отдела продаж — словом, от кого угодно, то эта информация не признавалась официально до момента её ввода в систему. С ростом компании большая часть работы по протоколированию ошибок ложилась на плечи службы технической поддержки, но идея оставалась той же.

Как далеко это заходит? Значит ли это, что если у одного разработчика возникла проблема с API, который реализовал другой разработчик, то её сразу же нужно запротоколировать? Вовсе нет. Разработчики могут решить проблему друг с другом самостоятельно, и тогда её не нужно протоколировать. Однако если решение проблемы займёт некоторое время и требует наблюдения (что особенно важно), то необходимо занести её в систему и описать, чтобы не забыть о её существовании. Эти правила распространяются на всех членов команды и на все отделы.

Другое преимущество от протоколирования неполадок — возможность отчитываться о них. Как здорово пойти на текущее совещание с полным списком всех крупных неполадок и людей, над ними работающих! Это эффективный способ фокусировать внимание в процессе разработки на важных вопросах. Заявивший об ошибке будет польщён тем, что её признали и она обсуждается. А тот, кто назначен для её разрешения, поймёт, что теперь не отвертеться.

Управление изменениями

Как было сказано в начале этого раздела, процесс управления изменениями может осуществляться при помощи системы устранения проблем и неисправностей. Так как все серьёзные проблемы запротоколированы, вы можете составить список необходимых исправлений. После того, как по конкретной неисправности принято решение, просто внесите информацию об исправлении в историю. Возможность пересматривать прошлые решения и причины их принятия — большой плюс. Это позволит избежать отговорок типа «я не помню» или «кажется, меня не было на том совещании». Все просто, понятно и под контролем.

Приоритеты на основе времени

Хотя классификация «низкий, средний и высокий» часто полезна при назначении проблемам приоритетов, всё же в одной категории оказываются десятки и даже сотни неполадок. Чего не хватает в таких приоритетах, так это элемента времени. Очень часто есть ошибки с высоким приоритетом, которые нужно срочно исправить к бета-версии, а есть такие (с таким же приоритетом), что могут подождать до последнего выпуска, поскольку они очень сложны или требуют дополнительного исследования.

Рассмотрите включение поля «Исправить к дате» для установления приоритетов на основе критерия времени. Значения, помещаемые в это поле, могут браться на основе внутреннего графика выпуска проекта, например, здесь может быть указан определённый этап, бета-версия или кандидат на выпуск. Чтобы определиться с приоритетом, задайте себе вопросы: «эту проблему нужно решить к этапу 2 или бета-версии 1?», «Что, если мы выпустим продукт сейчас, а ошибку исправим в следующем выпуске?»

С течением цикла разработки следует назначать для каждой ошибки конкретный срок исправления. Поступая так, вы без труда получите текущий список неисправностей для любого выпуска, в том числе для следующего.

Проверяйте и исправляйте ошибки

Одна из главных задач, выполняемых при контроле качества, — проверка того, что ошибки на самом деле исправлены. На этом этапе мы убеждаемся в том, что разработчик действительно осознал проблему и провёл достаточное количество испытаний. Когда ошибка исправлена и соответствующие изменения внесены в исходный код, разработчик устанавливает значение поля «Контролю качества: подтвердить» на «Истина», а статус — на «Ожидается процедура контроля качества». После того, как система контроля качества проверила исправление ошибки, тестировщик устанавливает её статус «Закрыто». Проблема может быть закрыта только системой контроля качества после соответствующей проверки.

Используйте замечания по выпуску

Когда мы сталкивались с неполадкой, которую следует описать в замечаниях по выпуску или файле Read Me, мы изменяли её статус на «Release Notes», но оставляли её открытой. В примечаниях по выпуску описываются известные проблемы, способы их обойти и информация, появившаяся в последний момент и не попавшая в официальную документацию. Когда наступал момент выпуска бета-версии или окончательной версии, было очень просто составить список проблем, о которых следует указать в замечаниях по выпуску. И только после того, как проблему разрешили, мы окончательно её закрывали.

Используйте стандартные запросы

Чтобы осуществлять полноценный поиск по базе данных проекта, важно иметь набор стандартных запросов. Эти запросы должны использоваться совместно и быть доступны всем членам команды; важно, что у каждого будет одинаковая картина данных. Хотя частные запросы хороши для редких и особых требований, их не следует использовать для распределения заданий членам команды. Риск неправильного составления запроса или указания неиспользуемого более поля достаточно велик. В таблице представлены наиболее важные запросы.

Все открытые ошибки

Позволяет менеджеру проекта и руководству оценить проект в целом

Все открытые ошибки для конкретного этапа

Позволяет команде увидеть, какие ошибки остаются открытыми в проекте для заданного этапа.

Все открытые ошибки для конкретного человека

Позволяет каждому человеку просмотреть свой текущий список ошибок

Все открытые ошибки для конкретного этапа и человека

Позволяет каждому человеку просмотреть свой список ошибок для заданного этапа.

Все открытые ошибки тестировщиков с полем «Контролю качества: проверить» = «Истина»

Позволяет команде просмотреть свой план тестирования

Все открытые ошибки с полем «Предложения» = «Истина»

Позволяет менеджеру проекта и руководству пересмотреть текущие предложения по изменениям

Другие способы применения

Далее приведены другие важные способы использования информации, попадающей в базу данных проблем и неисправностей. Эта информация поможет менеджеру проекта и руководству оценить мероприятия по проекту на макроуровне, а также другие проблемы, съедающие значительную часть времени.

В цикле разработки команда обрабатывает сотни, а может, и тысячи ошибок и проблем, и поэтому понимание того, что же всё-таки происходит с течением времени, может быть очень ценно.

Интенсивность возникновения и устранения ошибок

Интенсивность возникновения — это мера того, сколько новых ошибок или неисправностей было обнаружено за определённый период времени. Интенсивность возникновения взлетает вверх в начале проекта, но с течением времени снижается. Интенсивность устранения — это мера того, сколько ошибок или неисправностей закрыто за определённый период времени. Она снижается по мере устранения ошибок. Ниже показана интенсивность возникновения и устранения ошибок — для проекта эти данные могут быть весьма полезны (рис. 5-1 и 5-2).

Рис. 5-1. Интенсивность возникновения и устранения ошибок в начале проекта.

Рис. 5-2. Интенсивность возникновения и устранения ошибок в моменты, когда проект близится к завершению определённого этапа.

Как соотносятся интенсивность возникновения и устранения ошибок? В начале проекта вы столкнётесь с массой новых проблем, которые обнаруживаются (открываются) быстрее, чем устраняются (закрываются). По ходу работы интенсивность возникновения по отношению к интенсивности устранения перестанет расти и снизится, так как существующие проблемы будут закрываться быстрее, чем новые будут обнаруживаться. Особого внимания требуют резкие скачки, которые могут проявляться в определённый период. Рассмотрите проблему, зафиксированную в этот период, чтобы определить, не она ли породила ещё большее количество новых ошибок.

Обычно, когда близится завершение внутреннего этапа, выпуск бета-версии и кандидата на выпуск, интенсивность устранения выше интенсивности возникновения. Если это не так, то новых проблем появляется больше, чем решается, а это не то, что вы бы хотели видеть, приближаясь к периоду стабилизации или выпуску.

Интенсивность устранения поможет вам определить эффективность обнаружения причин возникновения неполадки, а также примерный срок устранения ошибок, которые могут появиться в будущем. Скажем, если интенсивность устранения в течение двух последних недель составляла 10 ошибок в день, это может быть большим успехом. Если у вас 100 открытых ошибок, то вполне закономерно ожидать, что все они будут устранены приблизительно в течение следующих 10 дней. Эта цифра конечно же не точна (для исправления какой-нибудь неполадки может потребоваться и неделя), но она позволяет понять, чего вам следует ожидать при наличии большого количества оставшихся ошибок.

Количество изменений

Количество изменений также может о многом поведать. Количество изменений показывает число обновлений информации о неполадке. Причина обновления не имеет значения. Большое число изменений — верный знак того, что не всё идёт так гладко. Так, оставшаяся неполадка может исследоваться многими людьми, и ни один из них не установит причину её возникновения. Возможно, её передавали из отдела технической поддержки к разработчикам, от тестировщиков — к разработчикам или между двумя разработчиками туда и обратно. Наблюдение за количеством изменений информации об ошибках поможет определить те, что требуют внимания со стороны ведущих специалистов и менеджера проекта.

Счётчик неудачных исправлений

Ещё один хороший способ оценки нестабильности проекта — счётчик неудачных исправлений для всех ошибок, которые считались исправленными, но на самом деле исправлены не были. При устранении неполадки от команды тестировщиков требуется подтверждение того, что ошибка действительно исправлена. Если проблема всё ещё существует или исправление не принято, ей возвращается статус открытой, а значение поля «Исправлено неудачно» устанавливается в 1. Если тестировщики снова не могут подтвердить устранения неполадки, значение счётчика увеличивается до 2 или 3. Это сигнализирует о серьёзности проблемы и говорит о необходимости вмешательства ведущих специалистов или менеджера проекта.

Дополнительные средства

Хотя средства управления исходным кодом и устранения проблем были стержнем процесса разработки в NuMega, мы регулярно использовали и другие инструменты.

Отладчики

Одним из наиболее важных инструментов для наших разработчиков был разработанный NuMega Technologies невероятно мощный отладчик для Windows — SoftICE. Он загружается при запуске системы как драйвер устройства на уровне ядра и предоставляет беспрецедентные возможности управления и наблюдения за внутренними процессами приложения и операционной системы. Команда разработчиков часто использовала SoftICE для разрешения наиболее сложных проблем при отладке.

Средства анализа производительности и полноты

Мы часто использовали средства анализа производительности, в том числе наши собственные продукты TrueTime и TrueCoverage, для настройки производительности собираемых приложений. Мы поняли, что эти инструменты нужно использовать регулярно в течение цикла разработки, а не в самом конце, когда времени на оптимизацию или устранение проблем не хватает. Анализ производительности проекта по завершении определённого этапа или при выпуске бета-версии предупреждает проблемы и часто раскрывает ошибки, которые могут не проявляться в коде до самого выпуска. Не ждите, пока вы

столкнётесь с проблемами производительности, а начинайте сразу применять средства анализа производительности.

Мы также поняли, что эти средства работают наиболее эффективно, если проект специально разрабатывается с учётом анализа производительности и полноты. Собирая такие приложения, мы интегрируем их с нашими тестовыми заданиями для раннего обнаружения проблем с производительностью или для лучшей оценки полноты наших тестовых заданий.

Средства написания сценариев и автоматизации тестирования

Так как наши планы тестирования требовали максимально возможной автоматизации, мы всегда интересовались инструментами, способными нам в этом помочь. Наиболее важными были средства написания сценариев (Perl, командные файлы DOS и т.п.) и средства автоматизации тестирования. В то время Visual Test был доступен по разумной цене и предлагался взыскательным тестировщикам и разработчикам. С его помощью мы тестировали пользовательский интерфейс, но в подавляющем большинстве мероприятий по автоматизации полагались на средства написания сценариев.

Из собственного опыта

Когда мы создавали группу разработки ПО, проверяющего исправность кода, члены команды не были до конца уверены в его значимости. Но с продвижением проекта, когда команда начала применять собственный продукт на собственном коде, стало ясно, что если разработчики блочно тестировали свой код с полнотой 80%, то качество продукта заметно повышалось.

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Проблемы с инструментами

• Отбор нужных инструментов

Одна из главных ошибок которую допускают команды разработчиков, — игнорирование нужных инструментов. Я убеждён в том, что система управления исходным кодом и система устранения проблем и неисправностей необходимы. В равной степени для команды разработчиков важны средства отладки, поиска ошибок, оптимизации производительности и проверки полноты. Они помогут решить многие уникальные проблемы, всплывающие на поверхность в процессе разработки.

Всё, что может ускорить и автоматизировать цикл разработки, критично для вашего графика. Слишком часто графики сбиваются, так как сложные ошибки или проблемы с производительностью вносятся в процессе разработки, но не выявляются на раннем этапе. Когда они обнаружены, людям без посторонней помощи устранить их практически невозможно. Времени искать нужные инструменты в Интернете просто не будет. Убедитесь, что определились с инструментами, которые вам понадобятся, интегрировали их в процесс разработки и обучили персонал работе с ними.

• Смена инструментов на средине пути

Одно из главных искушений в процессе разработки — замена того, что вы используете сейчас, новой версией или инструментом от другого производителя. Последствия такого решения обычно не просчитывают. Я не могу представить себе смены систем управления исходным кодом или устранения проблем и неисправностей без значительного сдвига графиков. Обычно лучше дойти до конечного выпуска с тем, что у вас имеется, нежели менять это на полпути.

Проблемы управления исходным кодом

• Структура проекта

С ростом проекта структура системы управления исходным кодом становится очень важной. Хотя здесь я предлагаю стандартный способ работы, не забудьте спланировать систему в соответствии с нуждами вашего проекта. Вы должны принять во внимание фактор одновременного выпуска нескольких версий (пакеты обновлений, сокращённые и полные выпуски), а также потребности разработчиков, тестировщиков, фактор обучения пользователей, человеческий фактор и технологические потребности.

• Содержание

Не обманывайте себя, думая, что исходный код — это всего лишь набор файлов, требующий контроля над изменениями. Как было отмечено ранее, управления требует великое множество файлов и документов — не ограничивайтесь преимуществами контроля над изменениями только для исходного кода. Даже если придётся переучивать людей, отвечающих за контроль качества или обучение пользователей, время будет потрачено не зря.

• Конфликты ключевых файлов

Типичный случай: разработчику X был выдан файл, и поэтому разработчик Y не может его взять для внесения важных изменений. Такие конфликты из-за файлов способны замедлить работу над проектом. Предусмотрите способ быстрого внесения критичных исправлений или изменений.

Лучший способ решить эту проблему — предотвратить её появление. Подумайте, можете ли вы разделить наиболее востребованную информацию на несколько файлов, основываясь на логических подсистемах, компонентах и классах. Разбивка содержимого файлов уменьшает вероятность конфликтов.

Когда разделить содержимое файлов невозможно, например из-за жёсткой связи между блоками, следует разработать политику, предусматривающую возврат файлов в течение определённого количества часов после выдачи. Также в случае недоступности файлов программисты могут работать над их копиями, а не над оригиналами. Ставший доступным файл можно взять на короткий срок, быстро обновить и возвратить на место.

В большинстве систем управления исходным кодом поддерживается слияние файлов. Это позволяет совместить изменения в файлах. Хотя обычно эта функция работает правильно, не позволяйте операцию слияния проводить автоматически. Вы должны убедиться, что проверили все изменения, сделанные в файле. Если в нашей системе управления исходным кодом поддержка слияния реализована плохо, можете воспользоваться такими редакторами кода, как Visual

Slick Edit и Code Write. Они помогут выявить различия визуально и проверить результат слияния перед его реальным осуществлением.

• Маркировка

Одно из главных достоинств системы управления исходным кодом — возможность маркировки набора файлов, включённых в выпуск. Не забывайте проделывать этот важный этап работы для каждого выпуска, в том числе на основных уровнях, по завершении этапов работы, при выпуске бета-версий и т.д.

Метки должны устанавливаться не только для файлов с исходным кодом. Помечайте файлы-сборки, установочные файлы, файлы документации, файлы контроля качества — словом, все файлы, включённые в выпуск. Метка должна быть информативной и следовать соглашениям об именах, принятым для проекта.

Из собственного опыта

Однажды у нас работала команда, которая оценивала свою систему поиска ошибок во время разработки. Спустя некоторое время они пришли к выводу, что им следует переключиться на использование нового продукта, так как в нём были реализованы новые возможности. Они запустили конвертор и загрузили ПО. К сожалению, через две недели работы с продуктом они обнаружили, что там нет поддержки некоторых отчётов, а производительность программы ужасна. Им нужно было вернуться к предыдущей системе, но так как в новой версии не было предусмотрено процедуры автоматического обратного перехода, им пришлось вручную водить все записи об ошибках и неполадках, внесённые с момента перехода.

Проблемы поиска ошибок и неисправностей

• Целостность данных

Обеспечение целостности данных в системе должно быть приоритетной задачей. Если вы не будете доверять информации в системе, то не станете её использовать, и она потеряет свою значимость. Очень важно определить правила обеспечения целостности данных, в большинстве основанных на внутреннем процессе разработки. Так, вы никогда не должны сталкиваться с ошибками, которые реально «закрыты», но для них установлен статус «исследуется». Чтобы отображать работу над ошибками, нужно со временем изменять статус ошибок. Также убедитесь, что вы вводите правильные значения в поля (информацию об этапе, информацию о выпуске и т.д.). Какими бы ни были у вас внутренние методы проверки целостности, не допускайте хранения недостоверных или устаревших данных, иначе команда разработчиков будет присваивать данным любые значения, а вся система перестанет внушать доверие и станет бесполезной.

Лучший способ избежать проблем с целостностью данных — это убедиться в том, что команда осознает важность этих данных и может обнаруживать и решать проблемы самостоятельно. Собственная мотивация заработает хорошо, если вы продемонстрируете реальную значимость этих данных для команды. Не забудьте также периодически пересматривать данные и обсуждать результаты с командой.

Я показал некоторые способы моделирования ключевых элементов цикла разработки с применением средств устранения проблем и неисправностей. Однако не стоит увлекаться и использовать всё, что только может подойти для вашей команды. Вместо этого определите ключевые потребности для процесса разработки и выберите простые средства для их реализации.

Основы системы контроля качества

Проблемы с контролем качества могут разрушить проект: сорвать сроки или испортить продукт так, что потребители не смогут им пользоваться. Какой бы ни была ваша компания — начинающей или транснациональной корпорацией, вы должны эффективно балансировать между качеством продукта и временем его представления на рынке. Успех или неудача зависят именно от этого.

В этой главе мы рассмотрим основы системы контроля качества в динамичной среде с ограниченными ресурсами, обычной для современных проектов по разработке ПО. Мы остановим своё внимание на том, что, когда, как и кто должен тестировать. Затем я продемонстрирую общее решение и расскажу о некоторых простых и эффективных приёмах управления тестированием.

Основные принципы

Начнём с принципов работы системы контроля качества. Лейтмотив — обеспечение качества непосредственно в процессе разработки. Продукту нельзя придать качество позже без значительных затрат денег, сил и времени. Создание качественного продукта основывается на четырёх простых принципах:

• тестирование продукта осуществляется параллельно с процессом его разработки;

• качество продукта улучшается регулярно при завершении каждого планового этапа разработки;

• тестирование необходимо максимально автоматизировать;

• качество является частью культуры и технологии.

Параллельное тестирование

В среде с ограниченным временем поиск и устранение проблем в кратчайшие сроки с момента их появления — условие необходимое. Раньше узнаешь о проблеме — раньше решишь. Ваша задача — тестировать функции программы сразу после их окончательного создания. Это и называется параллельным тестированием. Чтобы его правильно осуществлять, вы должны иметь средства автоматизированного тестирования или ресурсы для проведения ручных тестов, доступные в момент реализации новых функций. Если реализация функции запланирована на конец пятой недели, то команда испытателей должна быть готова протестировать её на шестой. Это правило следует применять ко всем основным функциям. Хотя автоматизированное тестирование предпочтительнее, к запланированному моменту должны быть готовы и ресурсы для ручного проведения этой операции.

Ниже представлен идеальный график параллельного тестирования набора функций программы (табл. 6-1). Заметьте, что разработка и тестирование идут максимально плотно друг за другом. Реализация функции завершается в конце недели, команда испытателей готова приступить к тестированию в начале следующей. Хотя разработчики ответственны и за создание кода, и за его базовое тестирование, команда тестировщиков в заданный период проверяет функцию по

максимуму. Так как разработчики и тестировщики должны работать над функцией вместе, их называют «оперативной командой».

Табл. 6-1. График работы оперативной команды над функциями A, B и C.

Разработчики и тестировщики несут обоюдную ответственность за своевременное обеспечение качества. В такой системе реализация функции не завершается написанием кода. Функция считается законченной, если она проверена тестировщиками и соответствует заданным критериям. Разработчики и тестировщики должны осознавать, что для завершения работы над функцией они должны работать вместе. У каждой группы свои задачи (написание кода, тестирование, автоматизация и т.п.), но чтобы сделать всю работу, они должны действовать сообща. Нельзя переходить к следующей функции, если текущий набор функций не проработан окончательно и не стабилизирован.

Запомните: одновременно должно разрабатываться функций не более, чем вы можете обеспечить их сотрудниками. Иначе говоря, число оперативных команд должно основываться на числе функций, для которых допустима параллельная работа (разработка и тестирование). Если тестировщиков больше, чем разработчиков, или наоборот, то при использовании такой модели разработки команда считается несбалансированной, и вам следует набрать дополнительный персонал в те области, где испытывается дефицит.

Наконец обратите внимание, что я добавил в график работ фазы стабилизации и интеграции. Эти фазы позволяют команде укрепить программу по завершении ключевых этапов, прежде чем продолжить работу над оставшейся частью проекта. Необходимость стабилизации и интеграции мы рассмотрим в следующем разделе. О том, как встроить периоды стабилизации и интеграции в график работ, см. главу 11.

Стабилизация и интеграция

Через каждые 4-6 недель команда должна отводить 1-2 недели (в зависимости от сложности проекта) на тестирование, стабилизацию и интеграцию функций, завершённых к данному моменту. Такие периоды стабилизации и интеграции идут на пользу команде, функциональности и качеству. Вы можете завершить незаконченное тестирование, начать интегральную проверку функциональности и определить проблемы, которые следует решить, прежде чем продолжить работу над проектом. Не обращайте особого внимания на мелкие неисправности и детали. Просто перед началом очередной стадии убедитесь, что структурно и функционально проект находится в хорошем состоянии. В течение этого периода все члены команды должны направлять свои усилия только на стабилизацию и интеграцию. Не работайте над новыми функциями, кодом или чем-то ещё до тех пор, пока вы не будете уверены в стабильности того, что уже построено.

Периоды стабилизации и интеграции также позволяют сопоставить фактическое продвижение проекта с запланированным. Если проект хорошо спланирован, вы будете точно знать, на какой неделе какая функция будет завершена. Укладываетесь ли вы в график? Можно ли использовать определённые

функции в намеченный срок? Эта информация необходима для того, чтобы не дать проекту выйти из колеи. Повторю, что о календарном планировании подробно говорится в главе 11, а сейчас просто запомните, что вам нужно заранее определить периоды, во время которых вся команда, работающая над проектом, будет концентрироваться на стабилизации и интеграции программы.

Автоматизация

Максимально возможная автоматизация процесса тестирования — ключ к параллельному тестированию (и раннему обнаружению ошибок). Автоматизация предоставит вам следующие преимущества:

• Сокращение внутреннего цикла тестирования

Автоматизация помогает выполнять тесты быстро. Для параллельного тестирования вы должны будете в течение всего цикла разработки иметь постоянную возможность тестировать большие части продукта за короткий промежуток времени. Ручное тестирование требует массу времени, больших трудовых затрат и не слишком надёжно. Его нельзя выполнять каждую ночь после очередной сборки. Автоматизация — единственный способ добиться максимальной эффективности от параллельного тестирования.

• Сокращение потребностей в персонале

Автоматизация значительно сокращает расходы на рабочую силу. Относительная стоимость выполнения автоматизированного тестового задания ничтожна по сравнению с ценой выполнения этой операции вручную.

• Проверка изменений, внесённых в последний момент

Изменения, которые вносятся в последнюю минуту, неизбежны, и чёткие автоматизированные тестовые задания незаменимы для быстрой проверки того, что эти изменения не приведут к серьёзным проблемам. Выполнять вручную все обязательные тесты после внесения лишь нескольких изменений дорого и порой просто невозможно.

• Обеспечение полноты тестирования для новых выпусков

С каждым новым выпуском команда должна быть уверена в том, что функции из предыдущих выпусков все ещё работают. Если вам снова предстоит вручную тестировать все функции из прошлого выпуска, у вас, возможно, не останется ресурсов для тестирования новых функций в том же объёме. С течением времени число тестов, которые нужно выполнить вручную, станет огромным. Автоматизация тестирования функций предыдущих выпусков поможет решить эту проблему.

Из собственного опыта

Однажды, в очередной раз сообщив боссу о продвижении проекта и заверив его в том, что всё в порядке, я установил сборку и нажал на кнопку «выполнить наиболее критичную функцию проекта». Она не работала. Выяснилось, что уже много дней сборка была сломанной, хотя большинство об этом не знало. По нашему графику мы уже давно прошли период разработки и тестирования этой функции, и коль уж однажды она работала как часы, то должна была работать и сейчас. Сейчас мы поняли, что если наиболее критичная функция продукта была нестабильна, то состояние остальных функций, которые тогда работали, сейчас

также неизвестно. Все были так заняты написанием нового кода и тестированием новых дополнительных функций, что никто не заметил того, что продукт больше не работал. Бета-версия была отложена на несколько недель.

В тот момент я и моя команда поняли всю важность автоматизации тестирования для нашего проекта. Мы начали писать автоматизированные регрессивные тесты для ключевых функций, запускать их каждый день и немедленно устранять серьёзные неполадки.

Чаще всего автоматизацию критикуют из-за времени, необходимого для создания хороших тестовых заданий. Да, тестовые задания требуют материальных и трудовых затрат, но, созданные на совесть, они приносят большие дивиденды. Я рекомендую выделить нескольких специалистов по автоматизации, чьей задачей в цикле разработки будет только написание автоматизированных тестовых заданий.

Время, необходимое для поддержания адекватности тестов будущим выпускам, также является объектом нападок. Особенно это относится к автоматизации тестирования пользовательского интерфейса. Если между выпусками в вашем пользовательском интерфейсе происходят серьёзные изменения, то тестовые сценарии могут не работать и потребовать больших усилий для их совершенствования. По этой причине при автоматическом тестировании следует сосредоточиться на функциях, не относящихся к пользовательскому интерфейсу. Автоматизируйте тестирование ключевых функций, а не деталей пользовательского интерфейса.

Отличная идея — строить продукт, изначально рассчитанный на тестирование. Если вы минимизируете свою зависимость от пользовательского интерфейса и создадите альтернативные способы ввода данных и просмотра выходных данных, то будете защищены от изменений в интерфейсе. Например, рассмотрите возможность использования файлов, записей реестра, параметров командной строки и COM-интерфейсов передачи входных данных. Для вывода данных вы можете использовать текстовые файлы, распечатку значений переменных или готовые компоненты, специально предназначенные для этой цели. Я не говорю о том, что пользовательский интерфейс не должен быть протестирован, — просто приоритетным должно быть автоматизированное тестирование ключевых функций продукта. Однако если вы решили автоматизировать тестирование пользовательского интерфейса, начните с «контактного» тестирования. При этом, чтобы проверить функциональность всего интерфейса, вызываются и закрываются все диалоговые окна.

Из собственного опыта

Работая в NuMega над BoundsChecker 5, мы знали, что команда, создающая внутренние компоненты, значительно опережает команду, занятую пользовательским интерфейсом. И мы должны были быть уверены в том, что сможем тестировать продукт, даже если у нас не будет пользовательского интерфейса месяц или больше. Команда, отвечающая за внутренние компоненты, разрабатывала простые драйверы, вызывавшие подсистему с данными, необходимыми для работы. Используя эти драйверы, мы могли тестировать

продукт и убедиться, что он твёрд, как скала, задолго до того, как пользовательский интерфейс был готов. Помимо раннего тестирования продукта, эти драйверы предоставляли надёжный и простой способ автоматизированного тестирования подсистем разных выпусков.

Команды, процессы и культура

У вас есть опыт создания качественного ПО? Ваши технологические процессы производительны и эффективны или они обычно занимают кучу времени и ресурсов? Учитывается ли вопрос качества каждым человеком, участвующим в разработке ПО? Как далеко вы готовы пойти, чтобы обеспечить качество? О качестве заботится каждый или есть такие, которые говорят: «это не мой участок»?

Эти вопросы могут определить, насколько группа успешна в разработке качественного ПО. Иногда говорят, что высшее руководство не желает брать на себя обязательства, необходимые для создания качественных продуктов. С другой стороны, они, может быть, и хотят поставлять качественный продукт, но не уверены в том, что у команды есть для этого эффективная система. Они считают, что увеличение количества процессов контроля качества всего лишь приведёт к дополнительным затратам времени и повысит расходы без улучшения продукта. Одной из задач этой книги является определение конкретного набора процессов контроля качества, которые позволят поставлять лучшие продукты в кратчайшие сроки, насколько это возможно. Однажды обзаведясь системой, в которой вы уверены, вы вероятнее всего станете поддерживать и постоянно использовать именно её.

Из собственного опыта

В NuMega менеджер проекта определял качество как главный приоритет для каждого члена команды. Он устанавливал продукт и использовал его почти ежедневно, фиксировал ошибки и обсуждал обнаруженные неполадки со своей командой на совещании, в обеденное время и встреч в коридоре. Это подгоняло всю команду, и каждый её участник включался в работу. Тестированием и оценкой результатов занимались все. Все осознали: ошибки — это зло, и с ними надо бороться. Мы давали всем понять, что до самого конца проекта о качестве будут помнить и не забудут после продажи продукта.

Что, когда и как тестировать

Тестирование эффективно, только если понятно, какую часть продукта, когда и как тестировать. Вроде вопросы простые, но если вы работаете в жёстком графике и с ограниченными людскими ресурсами, то вам нельзя терять время, тестируя объект слишком глубоко или, наоборот, слишком поверхностно. Нужно сосредоточиться на проверке в следующих ключевых областях продукта:

- процедуре установки;
- функциональных возможностях;
- интеграции функций;

• производительности.

Тестирование в этих областях должно происходить постоянно в течение всего цикла разработки. Однако для эффективного выполнения этой процедуры вам нужно знать, когда и как проводить тесты в каждой из этих областей. Короче говоря, для каждого крупного мероприятия в процессе разработки вы должны обладать набором хорошо определённых процессов и процедур, которые будут отлавливать проблемы. Эти процессы и процедуры источают в себя следующие компоненты:

• Входные тесты

Проверяют ПО перед подтверждением внесённых изменений.

• Ежедневные базисные тесты

Выполняются для каждой сборки программы.

• Тесты по завершении функции

Проверяют функцию сразу же после завершения работы над ней.

• Тесты при стабилизации и интеграции

Проверяется интеграция функций через определённые интервалы времени.

• Бета-тестирование и кандидаты на выпуск

Производится внешнее тестирование продукта через определённые интервалы времени.

В оставшейся части этой главы мы поговорим об этих пяти ключевых разновидностях тестирования.

Входное тестирование

Позволяет разработчикам проверить важные функции в локальной сборке перед помещением кода в основную базу. Хорошие тесты должны обладать:

• совместимостью между всеми машинами;

• простотой установки, запуска и выполнения;

• проверять ключевые функции или подсистемы продукта.

Входные тесты представляют наибольшую ценность для случаев, когда вы вносите исправления в критичную или сложную часть системы. Если входной тест выполняется неудачно, вы можете самостоятельно найти и устранить неполадку. Вы не нарушите работу остальных разработчиков, которые могут взять исходные файлы с ошибками, после внесения этих файлов вами в систему. Также входные тесты предотвращают внесение критических ошибок в ежедневную сборку и сбой базисного теста.

Ежедневное базисное тестирование

Ещё один способ реализации стратегии «тестировать как можно раньше», помимо входных тестов, — ежедневные базисные тесты. Так как вы строите продукт каждый день, то и тестировать его нужно ежедневно. Базисные тесты — это основной набор автоматизированных регрессивных тестов, проверяющих, что ключевые функции продукта работают. Они обеспечивают создание работоспособной сборки и гарантию того, что за прошедшие 24 часа не было значительных ухудшений. С добавлением новых ключевых компонентов базисные тесты также следует улучшать и расширять.

Из собственного опыта

Продукт BoundsChecker компании NuMega хорошо известен за свою способность находить утечки памяти в приложениях C/C++. Ежедневные базисные тесты для этой программы включали в себя приложение BugBench, в котором было множество утечек памяти, а также других отвратительных «жучков». Мы использовали эту программу-пример для генерации ошибок, которые BoundsChecker должен был уметь искать. Если BoundsChecker находил не все ошибки в программе-примере, тогда по определению сборка считалась плохой. Нам нравилось получать по утрам «ещё дымящийся отчёт о тесте» с результатами проверки сборки минувшей ночи. Такой процесс позволял нашему проекту почти всегда быть стабильным и работающим, поскольку наши базисные тесты сразу находили критические проблемы.

Заметьте: ежедневные базисные тесты не имели своей целью проверку незначительных функций, таких как проверка работоспособности предварительного просмотра перед печатью, или вызов справочной системы по нажатию клавиши H — все это легко проверить вручную. Мы концентрировались на ключевых функциях проекта.

Как и ежедневная сборка, данные о базисных тестах (предоставляемые в виде отчётов) совместно используются всей командой так, что каждому понятно, есть ли в продукте проблемы или нет. Если да, руководители разработчиков и тестировщиков должны провести расследование, быстро определить причину проблемы и назначить специалиста для её разрешения. Для этого специалиста разрешение данной проблемы должно быть наивысшим приоритетом.

Тестирование реализованной функции

Итак, ваша задача — тестировать каждую функцию, как только работа над ней будет завершена. Для каждой важной функции должны быть назначены разработчик и тестировщик, которые вместе будут отвечать за своевременную и качественную поставку этой функции. Такое назначение будет способствовать совместной работе, обмену информацией и идеями, а успех или неудача разделятся поровну. Для каждой существенной функции должны быть заготовлены автоматические тесты, но вы также должны быть готовы, если понадобится, протестировать их вручную. В вашем плане контроля качества должна быть изложена вся информация так, чтобы было абсолютно понятно, когда и как тестируется каждая функция.

Ключевые функции

Основные усилия команды, отвечающей за контроль качества, направляются на тестирование ключевых функций. Их можно тестировать как автоматически, так и вручную, но это надо делать сразу после того, как разработчик закончил кодирование. Чем раньше начать тестирование фикции, тем быстрее вы объективно оцените продвижение проекта и, если обнаружатся проблемы, начнёте их решать.

Тестировщики почти всегда будут находить проблемы. Для их устранения в графике разработки должно быть отведено некоторое время в период разработки компонента или в ближайшем периоде стабилизации. Я советую выделять немного времени в обоих периодах. Скажем, в пятидневном задании должен быть предусмотрен один день как раз для устранения неполадок, то есть 20% «лишнего времени».

Установка

К сожалению, процедура установки — самая «забытая» часть любого продукта. Люди редко думают о том, что установка — это важнейшая функция программы, и поэтому не уделяют ей должного внимания. Если вы не протестируете процедуру установки, можете пожалеть: этот компонент программы используют все. Единственный способ создать великолепное первое впечатление — это разработать отличную процедуру установки. Иначе пользователь с первых минут будет недоволен вашей программой.

Как и для остальных крупных компонентов, для проверки процедуры установки нужно выделить оперативную команду. То есть задача создания и проверки процедуры установки назначается технологам и тестировщикам. Эта задача должна входить в план проверки качества и выполняться регулярно в течение цикла разработки. Помните, что обычно установка — очень сложная часть программы, она требует безупречной работы на самых разных конфигурациях. И здесь автоматическое тестирование незаменимо.

Вот список основных тестов процедуры установки, которые должны быть выполнены для любого продукта, который вы собираетесь поставить.

• Операционные системы

Проверка на всех операционных системах, поддерживаемых вашей программой.

• Сервисные пакеты

Проверка со всеми сервисными пакетами ОС, поддерживаемых вашей программой.

• «Чистая» установка

Проверка установки продукта на ОС, где не установлены предыдущие версии программы.

• «Грязная» установка

Проверка установки продукта на ОС с установленными предыдущими версиями программы.

• Конфигурации продукта

Проверка поддержки процедурой установки различных конфигураций продукта.

• Функции программы установки

Проверка собственных возможностей процедуры установки (онлайновая регистрация, кнопка «Далее», кнопка «Назад», кнопка «Отмена» и т.д.).

• Тест удаления

Проверка процедуры удаления продукта.

Хотя хорошая процедура установки прежде всего предназначена для пользователей, вы тоже увидите, что она играет важную роль в ускорении работы по контролю качества. Так как команда тестировщиков должна работать с самой последней сборкой программы, у вас постоянно должна быть надёжная процедура установки, которую они будут использовать. Ведь вы не хотите, чтобы команда тратила время на редактирование реестра, копирование файлов, редактирование параметров конфигурации и т.д. Вам нужно направить их усилия на тестирование продукта, а не на ручные процедуры, в которых легко могут появиться ошибки.

Надёжная и простая в использовании процедура установки будет полезна для всех членов команды, а не только для тестировщиков. Каждый сможет установить продукт для своих собственных целей. Техническим писателям потребуется установка для создания описания функций продукта, разработчикам — для отслеживания «жучков», проблем с производительностью и оценки пользовательского интерфейса. Ваша команда должна работать с продуктом, а не бороться с его установкой.

Из собственного опыта

Не забудьте о процедуре удаления! В NuMega команды разработчиков и тестировщиков оценили значимость процедуры удаления. Ведь она позволяет получить чистую систему и не тратить время на ручное удаление записей реестра и файлов из системного каталога.

Тестирование при стабилизации и интеграции

До этого момента в цикле разработки тестирование было направлено на проверку отдельных функций. Но в период стабилизации и интеграции внимание уделяется:

• завершению всех отложенных тестов отдельных функций;

• проверке интеграции функций;

• проверке текущей производительности и нагрузки;

• исправлению всех серьёзных ошибок, изъянов проекта или архитектурных проблем;

• тестированию бета-версий и кандидатов на выпуск.

Завершение каждого из этих этапов очень важно для начала работы над следующей частью проекта. Давайте подробнее рассмотрим каждый из них.

Завершение тестирования отдельных функций

Задача номер один — завершение всех тестов, которые могли быть отложены. Это вполне нормально, когда какой-то оперативной команде для завершения всех тестов требуется больше времени, даже после 4-6 недель упорной работы. Используйте это время для выполнения всех тестов, которые ещё не были выполнены, так вы сможете поддерживать параллельное тестирование до самого конца проекта.

Проверка интеграции

Тестирование интеграции должно быть определено заблаговременно как часть плана тестирования. Лучший способ сделать это — создать набор примеров использования, предпочтительно с точки зрения пользователя, описывающих, как

различные функции должны работать вместе. Перед тестированием вы должны быть уверены, что заданные функции находятся в рабочем состоянии и в принципе могут использоваться вместе. Именно сейчас время их совместной проверки, и если они не станут работать, то настанет время исправления ошибок.

Тестирование производительности и нагрузки

Хотя конечный продукт нельзя оценить до тех пор, пока вся система не будет собрана воедино, для оценки прогресса или его отсутствия в цикле разработки могут проводиться наблюдения и предварительные измерения.

Не забудьте создать набор тестов, которые будут служить в качестве эталонных тестов для продукта, и выполнять их регулярно в цикле разработки. Выполняйте стрессовые тесты и оценивайте производительность системы по завершении определённых этапов и в моменты синхронизации и интеграции. Именно для этого разработка разбита на этапы. Выполнение тестов в эти моменты — лучший способ раннего обнаружения ошибок и их исправления до того, как вы продолжите строить ваш продукт на фундаменте, содержащем ошибку.

Из собственного опыта

Во время разработки BoundsChecker одной из главных проблем была производительность. Ничего не стоило полностью поменять характеристики производительности продукта, добавив несколько строчек кода в критичные функции. Чтобы обнаружить источник проблем с производительностью, мы использовали несколько тестовых приложений, которые нагружали BoundsChecker до предела. Одно из таких приложений называлось Torture. Оно создавало 256 параллельных потоков и запрашивало и освобождало десятки тысяч блоков памяти в куче. В течение всего периода разработки мы запускали Torture (и подобные программы), чтобы определить, не снизилась ли производительность продукта. Так как мы хотели видеть результат сразу, мы стали каждую ночь запускать Torture как часть наших автоматических регрессивных тестов и сравнивать показатели производительности. С таким уровнем контроля мы обычно могли определять снижение производительности в сборке предыдущего дня. Весьма неплохо!

Коррекция после тестирования

Период стабилизации и интеграции также позволяет исправить серьёзные ошибки до перехода к следующему набору функций. Тестирование функций и их интеграции выявит множество ошибок, а это именно то, что нужно. Вы сможете заранее устранить эти ошибки, что увеличит продуктивность дальнейшей работы. Но время на поиск и исправление этих ошибок должно быть учтено в графике.

Оценка после тестирования

Когда период стабилизации и интеграции подходит к концу, не забудьте оценить результаты и произвести изменения. Усовершенствовать ли аппаратную часть? Нужно лучше тестировать подсистемы? Требуется ли лучшее определение API? Больше людей? Какие бы изменения ни требовалось провести, это нужно сделать сейчас.

Это также подходящий момент решить, что имеет смысл улучшить во время следующего периода стабилизации и интеграции. Смотрите на фазу стабилизации и интеграции, как на проверку ПО и команды, которая его создаёт. Вы должны оценить производительность и программ, и людей и провести все необходимые изменения.

Пример тестирования

Рассмотрим простой пример, чтобы показать, как все эти области работают вместе. Допустим, вы создаёте Web-приложение и проходите фазу стабилизации и интеграции. Вы уверены в работоспособности определённых функций. Скажем, вам нужно только создание, редактирование и удаление покупателей. Но чтобы заставить эти функции работать, нужно потрудиться. Их работа затрагивает пользовательский интерфейс, Web-сервер, промежуточные звенья и базу данных. Все эти компоненты взаимосвязаны. В этом случае проверка интеграции может состоять из таких заданий:

- попытаться добавить покупателя;
- некорректное добавление покупателя (проверка полей);
- повторное добавление одного и того же покупателя;
- редактирование сведений о покупателе (всех полей, ни одного поля);
- редактирование сведений о несуществующем покупателе;
- некорректное редактирование сведений о покупателе;
- удаление существующего покупателя;
- удаление несуществующего покупателя.

Завершив тестирование интеграции, вы будете обладать сведениями о производительности приложения. Как долго добавить покупателя? А удалить? Получить сообщение об ошибке? Хотя может быть несколько причин плохой производительности, если вы не можете быстро добавить покупателя в маленькой базе данных, возможно, имеется проблема, требующая дополнительного исследования. Есть ли проблемы с драйверами БД? Может быть, у вас плохая структура БД? Нет ли претензий к промежуточному уровню? Чтобы это проверить, через определённое время нужно проводить мониторинг, устанавливать планку производительности для основных транзакций и регулярно сравнивать результаты, чтобы знать, что вы на верном пути.

Задача тестирования интеграции — убедиться в том, что к завершению первого этапа функциональность продукта удовлетворительна. Если это так, вы готовы приступить к следующему крупному этапу. Если нет, скажем, вы не можете добавить, отредактировать или удалить покупателя, остановитесь и исправьте всё, что мешает двигаться вперёд.

Тестирование бета-версий и кандидатов на выпуск

Тестирование бета-версий и кандидата на выпуск — ключевой этап проекта. Бета-тест — это возможность дать клиентам проверить и оценить ваше ПО за месяцы до его выпуска или применения в реальной рабочей среде. В большинстве проектов во второй половине цикла разработки предусматривается 2-3 бета-периода. Во время каждого такого периода привлекаются десятки или сотни, а

иногда тысячи пользователей. Кандидат на выпуск потенциально является последней сборкой продукта, и он ещё важнее. Если последний круг тестирования завершился без серьёзных проблем, то он представляет ПО, которое вы намереваетесь предоставить потребителям. (Подробно о бета-тестировании см. главу 13, о тестировании кандидатов на выпуск — главу 14.)

Одна из главных задач при работе с бета-версиями и кандидатами на выпуск — определить, что следует протестировать в сборке, прежде чем предоставить её сторонним организациям. Конечно, вы не сможете заново протестировать весь продукт. Полное тестирование и доводка продукта займёт месяцы, если не годы. Вместо этого вам нужно составить очень конкретный и хорошо продуманный план, который будет выполнен в очень сжатые сроки. (Для большинства небольших и средних проектов норма составляет 7-10 дней.) В планы тестирования бета-версий и кандидатов на выпуск нужно включить:

• выполнение всего набора автоматических тестов;
• выполнение набора ручных тестов, включая:
* нормальную установку/проверку лицензии (полностью вручную);
* тестирование базовых функций продукта (полностью вручную);
* тестирование производительности и нагрузки (полностью вручную);
* выборочную проверку на всех поддерживаемых платформах;
* другие специфические разделы проекта.

Этот список послужит вам хорошей отправной точкой, но для каждого пункта вы должны определить конкретный сценарий тестирования. И если все эти тесты будут успешно пройдены, вы выпустите вашу программу. Если вы не можете успешно выполнить тесты, устраните неполадки и повторите процесс.

Одна из черт грамотного цикла тестирования бета-версии или кандидата на выпуск заключается в его предеказуемости. Вы должны знать, сколько времени займёт выполнение автоматических и ручных тестов. Имея эту информацию, вы сможете точно предсказать, сколько времени займёт тестирование следующей бета-версии или кандидата на выпуск. Зная, сколько времени нужно для тестирования другой сборки, вы сможете оценить риск и стоимость внесения дополнительных изменений.

Кто должен тестировать?

За тестирование должны отвечать все участники проекта, невзирая на лица и отведённые им роли. При использовании продукта с любой целью и в любой форме делать это надо с критической точки зрения. Кем бы вы ни были: менеджером проекта, радостно рассматривающим новую функцию, автором руководства пользователя, проверяющим, как будет работать пример, или специалистом по инженерной психологии, устанавливающим продукт для проверки пользовательского интерфейса — вы должны отслеживать, искать и сообщать о проблемах качества.

Имея сжатые сроки и ограниченные ресурсы, трудно ожидать, что одна группа сможет провести всю работу по тестированию, особенно если учитывать, что в командах тестировщиков дефицит кадров проявляется чаще всего. Так что

убедитесь в том, что ваши разработчики, технические писатели, инженерные психологи, менеджер продукта, менеджер проекта, вице-президент или студенты, проходящие летнюю практику, будут искать проблемы каждый раз, когда они используют продукт для своих нужд. Любой ценой заставьте их сообщать о найденных проблемах.

В период стабилизации и интеграции к тестированию приступает вся команда разработчиков — это коллективная работа. Появляется шанс увидеть, где команда находится в данный момент и сколько ещё нужно сделать. Обычно руководитель группы контроля качества выполняет задачу, распределяя ответственность за тестирование между всеми членами команды. Большую часть времени работа проводится в областях, где автоматические тестовые задания справляются плохо. Также сотрудников просят «сыграть роль пользователя» для ключевых частей продукта. Итак, команда, работавшая над продуктом в течение всего цикла разработки, просто незаменима. Это то, что должно стать частью вашей культуры и одним из ваших основных технологических процессов.

Эту идею можно развить ещё дальше — самим использовать собственные программы. Такой подход называют «питаться кормом своей собачки», он хорошо известен в нашей отрасли и может оказаться очень ценным. Для определения и разрешения проблем с качеством не нужно делать ничего, кроме как задействовать свои программы в реальной работе. Даже если в рамках вашей команды разработчиков продукт применить нельзя, попробуйте попросить нескольких опытных пользователей поэкспериментировать с программой. То, что они найдут, может оказаться для вас сюрпризом.

В определённый момент крайне необходимо чётко разграничить обязанности тестировщиков от обязанностей других членов команды (прежде всего разработчиков) в том, что касается тестирования. Чтобы люди концентрировались на своих прямых задачах, необходимо разделение труда.

Разработчики влияют на качество продукта больше всего. В конце концов они находятся ближе всего к коду, и риск внесения ошибок исходит прежде всего от них. Чтобы гарантировать отлов «жучков» до того, как команда тестировщиков увидит функциональный блок, они должны его тестировать в процессе написания. Хороший разработчик ускорит работу тестировщиков, предоставляя им надёжные функции. И наоборот, плохой разработчик затормозит работу тестировщиков, выдавая им компоненты с таким количеством ошибок, что о тестировании уже и речи не будет. Для тестировщика нет ничего более неприятного, чем находить массу очевидных проблем, которые разработчик мог найти сам всего за несколько минут работы.

Из собственного опыта

В NuMega мы готовили вторую бета-версию BoundsChecker 3. Для оценки продвижения проекта мы устраивали ежедневные совещания. Кэрол, наш ведущий специалист по контролю качества (в то время команда тестировщиков состояла из неё одной), настойчиво повторяла, что сборка была крайне неудачной.

Она сказала, что больше не будет зря тратить время на её тестирование и останется дома до тех пор, пока разработчики не приведут все в порядок, и быстро ушла.

Я готов был зааплодировать. Не потому, что мне нравилось состояние бета-версии. Кэрол дала понять разработчикам, что в их обязанности входит базовое тестирование программ и самостоятельная работа над проблемами кода. До команды разработчиков это дошло. Мы согласились и занимались тестированием и исправлениями в программе, пока не почувствовали, что готовы позвать Кэрол. Это заняло около двух дней.

В отношении тестирования разработчики имеют ряд обязанностей:

• анализ плана тестирования;

• тестирование на уровне модулей (работает ли функция в большинстве ситуаций);

• предварительное интегральное тестирование (работает ли функция в связке с другими);

• протоколирование или устранение всех неполадок, обнаруженных в программе, когда они сами её использовали.

Команда, отвечающая за контроль качества, пропускает эту простейшую работу. Считается, что тестирование на таком уровне полностью проведено разработчиками до передачи функционального блока тестировщикам. Конечно же, тестировщики не слепо верят в то, что все абсолютно верно, они просто предполагают, что качество продукта находится на уровне, достаточном для того, чтобы приняться за свою работу.

Далее команда, отвечающая за контроль качества, проводит тестирование продукта на другом уровне. Она сосредоточивается на:

• планировании тестов;

• автоматизации создания тестов;

• автоматизации тестирования;

• тестировании функций в различных комбинациях;

• тестировании процедуры установки;

• тестировании интеграции и связи с системой;

• тестировании производительности и нагрузки;

• ручном тестировании (функций, для которых неприменимо автоматическое тестирование);

• диагностике проблем и их протоколировании;

• проверке исправлений и «закрытии» ошибок.

Хотя все эти обязанности привычны для тестировщиков, последняя может быть менее знакомой. «Закрытие» ошибки должен проводить только тестировщик — член оперативной команды. Задача разработчика — исправить ошибку в коде, занести исправление в систему управления исходным кодом и обновить статус ошибки на «Исправлено» в системе устранения неполадок. Но пересмотр всех исправленных ошибок и проверка того, что они действительно исправлены,

входит в обязанности тестировщиков. Только после такой проверки ошибка считается официально «закрытой».

Другие критичные моменты для контроля качества

Почти каждая команда столкнётся с рядом других вопросов. Это проблема тестирования на разных платформах, должная роль и использование ручного тестирования, а также инфраструктура, отвечающая потребностям проекта.

Матрица тестирования

Одной из функций контроля качества, занимающей массу времени, является тестирование продукта на широком спектре конфигураций ПО. Сегодня большинство продуктов поддерживают работу под управлением нескольких ОС в различных конфигурациях. Тестировать продукт на всех (если речь идёт о ручном тестировании) — гиблое дело.

К счастью, существует способ здорово облегчить эту задачу. Если у вас есть надёжные автоматические тестовые задания для проверки важнейших функций, можете задействовать их все для всех конфигураций, которые решили поддерживать.

Из собственного опыта

В NuMega мы решили проблему тестирования на нескольких конфигурациях путём распределения их между разработчиками и тестировщиками. Один получил Microsoft Windows 95, другой — Microsoft Windows 98, третий — Microsoft Windows NT 3.51 и ещё один — Microsoft Windows NT 4.0. От каждого требовалось выполнить тест на своей ОС в надежде как можно раньше — в процессе разработки — обнаружить проблемы. Таким простым способом мы почти сразу находили проблемы на всех платформах.

Ручное тестирование

Я столько внимания уделил автоматическому тестированию, что у некоторых из вас мог возникнуть вопрос: стоит ли вообще использовать ручные тесты? Да, но нужно понимать, где их применять. Ручные тесты используются в следующих случаях:

• Для тестирования ключевых функций в случаях, когда автоматические тесты запаздывают или вовсе не существуют

Что, если у вас нет времени или ресурсов для написания всех автоматических тестов, а команда разработчиков уже выдаёт вам готовую функцию? В таком случае нужно приступить к ручному тестированию, чтобы оценка функции проходила согласно графику. Раннее обнаружение ошибок и их устранение остаётся главной задачей.

• Для редко изменяемых, некритичных функций

Иногда значимость автоматического тестирования проигрывает простоте и быстроте ручных тестов. Если небольшую функцию легко протестировать и в ней не предвидится изменений, лучше пропустить автоматические тесты и направить свои усилия на более серьёзные задачи.

• Когда все трещит по швам

Когда сроки поджимают, а вам нужно быстро провести массу тестов, многие любят приглашать дополнительных испытателей, часто это оказываются люди, у которых опыт работы с продуктом небольшой или отсутствует вовсе. Для эффективного выполнения такой задачи следует иметь чёткий план ручного тестирования. В нём нужно описать важнейшие части продукта, которые требуется обследовать, и те моменты, которые нужно проверить наиболее тщательно. Это позволит просто распределить обязанности по тестированию всего продукта, и вы будете уверены, что самые критичные части продукта вошли в планы тестирования.

Но помните: нельзя быть зависимым от ручного тестирования. Его наращивание потребует больших затрат, а тестирование параллельно с разработкой продукта становится затруднительным.

Оборудование для тестирования

В проектах с жёсткими временными рамками нужно быть уверенным, что работа команды не замедляется из-за недостатка элементарного аппаратного или программного обеспечения. В разных командах и проектах требования к оборудованию будут заметно меняться, поэтому ниже перечислены основные требования к оборудованию.

• 2-3 компьютера на каждого тестировщика

Один будет использоваться для производственных нужд: электронной почты, отчётов о неполадках, автоматизации разработки и т.д., а остальные для тестирования. Нужно иметь возможность в любой момент менять конфигурацию тестовых компьютеров. Хорошо, если один из них представляет машину конечного пользователя.

• 2 компьютера на одного разработчика

Помните: разработчики тоже занимаются тестированием. Один компьютер им нужен для разработки, другой — для тестирования. Разработчики могут переконфигурировать его при «охоте на жучков», и это не помешает их основной работе. Повторю: хорошо, если одна машина представляет компьютер конечного пользователя.

• Доступная библиотека программ

Все ПО, которое требуется для разработки или тестирования, должно быть постоянно доступно. Для быстрого и простого доступа сотрудников к инструментам, продуктам и ОС, необходимым для работы, удобен дисковод с автоматической сменой компакт-дисков. Конечно, придётся позаботиться о наличии лицензий, но избавление сотрудников от хождения по коридорам в поисках нужного диска того стоит.

• Тестовая лаборатория

Великая вещь! Стойка с тестовыми компьютерами, на которых установлены различные ОС, с различными языками и сервисными пакетами здорово упростит работу по контролю качества для всей команды. Тестовая лаборатория хороша и для установки сложной среды тестирования, сборка и настройка которой отнимает массу времени.

Конечно, следование этим рекомендациям увеличит расходы на аппаратное и программное обеспечение, но дополнительные расходы обернутся приростом производительности и качества.

Из собственного опыта

На заре NuMega у нас не было постоянно доступной библиотеки программ, а охота за компакт-дисками здорово раздражала и отнимала драгоценное время. Часто наши планы требовали поддержки самой последней ОС или компилятора Microsoft. К счастью, мы участвовали в тестировании их бета-версий и регулярно получали обновления. Жаль, что только на одном компакт-диске. Когда кому-то требовалась последняя бета-версия Windows или Visual Studio, начиналась охота за диском. Если везло, мы находили человека с диском, который нам требовался, но чаще всего мы слышали: «Я отдал его тому-то», — и продолжали идти по следу. (Однажды я ходил так от одного к другому и только пятый человек в цепочке сказал мне, что этого диска в глаза не видел!) Если такой способ не работал, мы писали сообщение по электронной почте и с надеждой ждали ответа, а это время занимались чем-то другим.

После того, как в течение нескольких месяцев мы столкнулись с десятками таких сообщений, мы окончательно поняли, что проблему нужно решать, тем более что наша компания росла. Решением стала «вертушка» компакт-дисков. Это сработало, но только после того, как мы перевели все в режим онлайнового доступа. Наши попытки создать традиционную библиотеку не увенчались успехом, так как люди, бравшие компакт-диски, никогда не возвращали их на место, и мы вновь задавались вопросом: «У кого диск?»

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Нехватка ресурсов

Нехватка ресурсов (здесь я имею в виду ресурсы человеческие), вероятно, является наиболее частой проблемой системы контроля качества, и, честно говоря, она гораздо сложнее, чем может казаться. Если для контроля качества у вас нет необходимых ресурсов, прежде всего определите, в чём проблема. Если вам постоянно не хватает ресурсов для осуществления контроля качества, а рабочие места остаются вакантными, значит, вы испытываете проблемы с набором персонала, обратитесь к главе 1 за дополнительными разъяснениями. Если сотрудники, отвечающие за контроль качества, из-за дополнительной работы или сокращения графиков уже работают на износ, стоит рассмотреть возможность привлечения контрактников. Однако, прежде чем пойти на этот шаг, у вас должны быть полностью готовы планы тестирования. Важно, чтобы временные сотрудники выполняли план, а не писали его.

Если работы просто больше, чем ваши сотрудники могут выполнить, а вы хотите поставить качественный продукт, существует только два выхода:

• пересмотреть графики, чтобы они отвечали ограничениям, накладываемым разрабатываемыми функциями и возможностями персонала;

• пересмотреть функциональность программы, чтобы она отвечала ограничениям графика и возможностям персонала.

В первом случае вы распределяете работу по контролю качества между членами команды. Это обычно отодвигает сроки, так как каждому приходится выполнять дополнительную работу. Однако вы знаете, что держите планку качества и в то же время обеспечиваете работу персонала, следуете графику и реализуете нужную функциональность. Прежде чем сделать такой выбор, обратите внимание на командный дух, сроки и текущее состояние дел, а также последствия задержки выпуска.

Во втором случае вы сохраняете график (что часто очень критично) и качество продукта (что не менее важно). Причина, по которой этот путь является успешным, заключается в том, что общая нагрузка на всю команду и общий риск проекта снижаются. Поскольку исключённые функции не нужно разрабатывать, тестировать и описывать, производство продукта идёт быстрее. Прежде чем пойти на такой шаг, внимательно изучите функции и их важность для успеха продукта. Я пришёл к выводу, что лучше раньше выпустить продукт с несколькими хорошими функциями, чем поздно поставить то же самое, но с дополнительными возможностями. (В главе 11 я расскажу о приоритетах в выборе функций в подобных ситуациях.)

Недостаточная подготовка

Многие проекты «встают не с той ноги» и, честно говоря, обречены с самого начала, так как члены команды просто к ним не готовы. У вас должны быть основные планы, средства автоматизации и оборудование, о которых говорилось выше. Все это потребуется почти с самого начала разработки. Если вы будете писать планы или ждать поставки нужного оборудования в процессе разработки, вы уже опоздаете и не сможете делать то, что от вас требуется — тестировать.

После того, как масштаб необходимых ресурсов для осуществления контроля качества становится понятен, команды часто начинают рассматривать возможность добавления ресурсов в проект. Если это сотрудники, работающие по контракту или перешедшие из других отделов, то скорее всего у них не будет специальных знаний о самом продукте. Они не смогут применять автоматические тесты (возможно, потому что ни одного не будет написано) или выполнять ручные, так как у них не будет контрольного списка или материалов, описывающих, что следует проделать. В этом случае лучший способ продвижения вперёд — заставить их «играть пользователей». Хотя такой подход часто даёт неплохие результаты, не злоупотребляйте им или используйте его как замену способов тестирования, описанных в этой главе.

Отсутствие автоматизации

Надеюсь, к данному моменту стало абсолютно понятно, как важны автоматические тесты в работе по контролю качества. Без автоматизации объём ручной работы и количество персонала взлетят до небес, что заметно сдвинет ваши графики. Очень важно, чтобы команды, отвечающие за контроль качества, и разработчики писали так много автоматических тестов, как это возможно, и, конечно, не меньше, чем описано в рекомендациях, приведённых мной.

Ненадлежащее исполнение обязанностей

Проблемы с качеством не всегда являются результатом игнорирования приёмов и концепции контроля качества. Это может быть следствием ненадлежащего исполнения обязанностей. Если вам приходилось беседовать с менеджерами или ведущими специалистами о контроле качества в таком проекте, они, вероятно, постарались наговорить много всего о том, что нужно сделать. Но когда вы видите их проекты, то замечаете, что ничего не делается. Создание качественного продукта требует усилий: сосредоточенности, активного участия, исполнительности. Это не теоретические выкладки — все члены команды должны действовать активно и увлечённо.

Неправильная расстановка акцентов

Я настоятельно рекомендую тестировать продукты сначала вширь, а затем вглубь. Убедитесь, что все основные функции реализованы и нормально работают, прежде чем тратить время на второстепенные функции. Конечно, как я говорил ранее, следует расставить приоритеты в тестировании функций. Однако очень часто команды тратят слишком много времени на мелкие детали какой-то функции, в то время как оставшаяся часть продукта разваливается. Возьмите в качестве примера постройку здания. Какой смысл полировать все до блеска в вестибюле, когда лифты не работают!

Основы технологии разработки программ

Сборка и установка ПО — постоянно усложняющаяся задача. По сути, она стала настолько трудоёмкой, что для её решения возникла особая дисциплина — технология разработки законченного программного продукта. Эта технология является решающей для своевременного выпуска продукта.

Какой бы ни была ваша организация, большой или маленькой, вы должны иметь возможность на регулярной основе собирать и устанавливать ваше ПО. Однако слишком часто команды разработчиков неделями и даже месяцами не могут собрать или установить свою собственную программу. Хуже того, никто из них не отвечает за проблемы со сборкой и процедурой установки, так что эта проблема тормозит процесс разработки. Из-за того, что проект невозможно собрать или установить, могут появиться проблемы любого рода, что вызовет задержки. Если вы не знаете точно реальное состояние вашей программы, потому что не можете увидеть или использовать её, значит, вы действуете вслепую.

Технологи по разработке ПО

Это члены команды, работающей над проектом, которые имеют необходимые навыки работы с процессами и технологиями сборки и установки ПО. Хотя технологи могут выполнять множество обязанностей, в контексте нашего обсуждения выделим наиболее важные:

• определение, создание и сопровождение сборочной среды продукта;

• определение, создание и сопровождение процедуры установки продукта;

• определение, создание и обслуживание пакетов исправлений или сервисных пакетов;

• проведение модульного тестирования и основных мероприятий по контролю качества процедуры установки;

• разработка инструментов, сценариев и автоматизация разработки ПО;

• планирование сборочной среды (сборочной лаборатории).

Для выполнения этих задач технологи должны быть включены в команду, работающую над проектом, с самого начала до конца. Они должны создать план сборки и процедуры установки в соответствии с требованиями проекта, как они понимаются в настоящий момент. Им следует участвовать в совещаниях по проекту точно так же, как и остальным членам команды.

В небольших группах отдельный постоянный технолог не нужен. Вместо него эти обязанности могут выполнять другие члены команды по совместительству. Но со временем сложность ПО и размер команды разработчиков возрастают, и потребуются отдельные технологи. А ещё позже — централизованная структура, занимающаяся технологией создания готового продукта. Не надо предполагать, что эта функция не важна или её

качество не имеет значения только потому, что в начале её выполнение не потребует работы с постоянной занятостью.

Сборки

Сборка является результатом компиляции всего исходного кода продукта. Для корректного построения вашего ПО, интеграция должна быть обеспечена на самом элементарном уровне — на уровне исходного кода. Целостность исходного кода должна быть совершенной: ошибки компиляции и компоновки недопустимы. В сложных проектах совершенства добиться тяжело из-за массы связей между модулями исходного кода. Однако регулярно собирать свою программу можно и нужно.

Почему они важны

Способность собирать ПО является определяющей для поставки программ в срок. Одна из наиболее часто возникающих проблем при создании ПО — заставить все части работать вместе. Если о ней забыть до окончания проекта, то потом решение проблемы займёт недели или месяцы работы. В худшем случае потребуется переопределение каких-то API и функций. А это, естественно, означает появление никем не запланированных задержек.

При регулярном создании сборок разработчики могут проверять интеграцию кода. Интерфейсы API, файлы заголовков, параметры, типы данных и макросы — все должны быть в полностью рабочем состоянии, иначе сборка пройдёт некорректно. Сбой при сборке заставит разработчиков общаться друг с другом и при необходимости изменять программу. Но ведь это именно то, что вам нужно: искать и устранять проблемы на раннем этапе, а не в самом конце, скажем, за день до того, когда от вас требуется бета-версия.

Как их создавать

Далее приведён ряд рекомендаций о том, как сделать задачу создания сборки более простой и эффективной.

Утилита Make

Поддерживает набор правил сборки и отношений в программе для всего приложения или компонента. Описав эти правила, Make может решить, какие образы необходимо собрать и какие исходные файлы должны быть откомпилированы или скомпонованы.

Make существует уже несколько десятилетий, всё началось в UNIX, а затем она появилась практически на всех остальных платформах. В течение многих лет её улучшали, и последняя версия — Nmake — входит в состав Microsoft Visual Studio. Обязательно изучите утилиту Make в вашей среде разработки и используйте её для автоматизации задач сборки ПО.

Номера сборок

Разработчики используют номера для уникальной идентификации сборок. Номер сборки — это монотонно возрастающая целая величина, ни разу не повторяющаяся в истории создания приложения. Номер увеличивается на базовых уровнях, этапах и в каждом последующем выпуске ПО. Когда сборка приложения происходит просто, в вашей среде разработки и тестирования, вероятно, будет большое число разных сборок. Со временем возможность идентификации определённой сборки, установленной на машине, а также программных компонентов, сопровождающих её, становится очень значимой. Также это относится к идентификации сборок, в которых появились или были устранены крупные неисправности. После того, как программа выпущена для потребителей, возможность идентифицировать определённую сборку станет ещё критичнее.

Номер увеличивается на единицу каждый раз при создании очередной сборки. Обычно увеличение номера происходит в самом начале процедуры сборки, затем он помещается в рабочие файлы, и все компоненты могут включать его в свой состав или ссылаться на него. Обычно номер сборки указывается в окне, вызываемом командой About меню Help, так что все пользователи могут видеть, с какой сборкой работают.

Сборочные машины и лаборатории

Сборочная среда — это набор приложений, инструментов, библиотек и компиляторов, нужных для компиляции и компоновки ПО. Часто лучше всего установить эту среду на нескольких выделенных сборочных машинах, которыми распоряжаются и управляют исключительно технологи по разработке ПО, изменения на этих машинах недопустимы. Важно обеспечить и регулярное резервное копирование дисков этих машин, чтобы восстановление было простым и быстрым. А чтобы избавиться от неожиданных трудностей, не забудьте установить антивирусное ПО.

С ростом числа ваших сборочных машин потребуется целая сборочная лаборатория. Лаборатория полезна, когда нужно параллельно собирать действительно большие программы или большое количество программ (возможно, по ночам). Сборочная лаборатория поможет обезопасить наши машины и предотвратить посторонние вмешательства, способные привести к сбою.

Оповещение и сбои

О завершении сборки команду надо оповестить. Извещение может быть послано в список рассылки всей команды проекта или для этих целей может быть создан свой список рассылки.

Оповещение всех участников команды особенно ценно, если появляется сбой. Когда такое происходит, очень важно, чтобы ведущий разработчик или сотрудник такого же уровня посмотрел журнал ошибок и определил природу проблемы. Он будет отвечать за решение проблемы до тех пор, пока для её решения не будет назначен конкретный специалист.

Проверка

Созданная сборка должна быть помещена на сетевой диск с совместным доступом, где она будет проверена при помощи автоматических тестов, созданных командой тестирования. Проверка — очень важный этап, так как наличие готовой сборки ещё не означает, что продукт в рабочем состоянии. Вы знаете только то, что можете компилировать, и компоновать все нужные файлы.

Один из лучших способов проверить сборку — установить продукт и запустить базисные тесты. Для эффективного управления этим процессом для сборок следует завести два каталога.

• Самая последняя сборка (MRB)

В этом каталоге хранится самая последняя сборка программы. Однако она может и не устанавливаться или не работать правильно.

• Последняя хорошая сборка (LKGB)

Здесь хранится последняя хорошая сборка. Убедившись в том, что текущая сборка находится в хорошем состоянии (она установилась и прошла базисные тесты), скопируйте содержимое каталога MRB в каталог LKGB.

Частота сдачи и проверки сборки

Чтобы не сломать сборку, ответьте на следующие вопросы.

• Когда я должен сдать мой код?

Сдавайте свою часть кода, когда у нас есть что добавить к проекту. Это может быть совсем простое добавление, скажем, набор заглушек API, или очень сложное, например, крупный компонент. Но вы должны сдавать свой код часто. Смысл в том, чтобы как можно раньше заставить работать код, созданный разными людьми.

• Как я могу быть уверен в том, что не испорчу сборку?

Если возможно, для проверки кода осуществите локальную сборку программы. Вы как обычно берете код из системы управления исходным кодом, интегрируете ваш код и возвращаете его в систему. В большинстве проектов эта процедура выполняется просто, и это отличный способ гарантировать то, что вы не испортите сборку. Дополнительно, чтобы убедиться в отсутствии новых ошибок, можно запустить входные тесты.

Процедура установки

Процедура установки важна не только для потребителя проекта. Здесь я расскажу, почему процедура установки так важна и как строить процедуру установки параллельно разработке ПО.

Почему это важно

Процедура установки служит для выполнения двух важных функций. Во-первых, она заставляет команду думать об установочной среде, которая

требуется для продукта. Процедура установки требует от вас знания состава приложения: образов, библиотек, компонентов, файлов справки, библиотек типов и т.д. Также она заставляет вас определить исполняющую среду, в том числе поддержку драйверов баз данных, стандартных компонентов и операционных систем. Если вы сохраните компоненты продукта целыми и актуальными, вы сможете избежать проблем в дальнейшем.

Во-вторых, при наличии процедуры установки у членов команды имеется простой доступ к самым последним сборкам программы. Им не нужно запоминать все ненужные подробности по поводу установки программы, такие как местоположение файлов, процедур регистрации компонентов, команд запуска, параметров реестра и т.д. Они могут просто установить продукт и использовать его для своих целей. Примеры использования перечислены далее.

• Разработчики смогут увидеть свои компоненты со стороны официальной сборки и оценить проблемы, используя ту же процедуру установки, что и вся команда.

• Тестировщики будут устанавливать программу обычным образом и тестировать её на наличие проблем. Работать с последней сборкой будут как автоматические регрессивные тесты, так и вся команда, которая будет тестировать последнюю хорошую сборку. Это обеспечивает тестирование самой последней и наиболее стабильной версии программы. Единая официальная сборка упрощает и определение работоспособности компонентов. То, что разработчику удаётся заставить компонент работать на своей машине, не имеет значения, если компонент не работает в официальной сборке. Если в официальной сборке компонент, установленный при помощи текущей процедуры установки, не заработал, значит, он не работает вообще.

• Для корректного составления документации техническим писателям нужно видеть, использовать и оценивать программу. Доступ к сборке, которую можно установить, заметно ускоряет их работу, так как новые возможности, добавленные разработчиками в сборку, видны и могут быть документированы на следующий день.

• По мере развития сборки специалисты по инженерной психологии смотрят за тем, как пользовательский интерфейс продукта претворяется в жизнь, оценивают его и дают рекомендации. Без официальной сборки у психологов нет простого доступа к компонентам, с которыми они должны работать. В итоге проколы и несогласованности проекта обнаруживаются в процессе разработки слишком поздно.

• Значительно расширяются возможности менеджера проекта. Наличие официальной сборки обеспечивает отличное видение текущего состояния проекта. Состояние компонентов, параметров производительности, качества онлайновой справочной системы и т.д. перестаёт быть секретом.

• И, наконец, имея процедуру установки на раннем этапе, расширяется обратная связь с другими группами, такими как менеджеры продукта, специалисты по технической поддержке и отдел продаж. Каждая из этих

групп даст ценные отзывы о продвижении продукта, а также сможет отловить несколько ошибок.

Как её создавать

Хотя конкретные детали по применению значительно отличаются для разных продуктов и приложений, подход к процедуре установки всегда одинаков. Вы начинаете создание процедуры установки в самом начале проекта и со временем наращиваете её.

Скелет

Первый шаг в создании процедуры установки — конструирование скелета. Задача проста: сделать так, чтобы первый набор файлов был скопирован в каталог установки, Если даже программа не может сделать ничего, кроме как вывести на экран надпись «Hello World», для неё нужно создать процедуру установки. Она не должна быть сложной, но вам по крайней мере следует создать инфраструктуру, на основе которой начнётся строительство.

Мышцы

С продвижением проекта строительство продолжится на основе простой структуры, созданной вами, путём добавления сложных и утончённых элементов. Смысл в том, чтобы улучшать процедуру параллельно разработке проекта, т.е. сначала вы строите скелет, а со временем наращиваете его. Скажем, завершены новые компоненты, включена поддержка новых ОС или баз данных, упрощён текст лицензионных требований — вам нужно добавить новые файлы и изменить процедуру согласно новым требованиям.

Следует добавлять компоненты в процедуру установки лишь по необходимости. Ваша задача — написать план разработки процедуры установки, обеспечивающий включение определённых компонентов и поддержки, необходимой для разработки и тестирования. Этот план должен помочь вам найти равновесие между двумя крайностями: ежедневного внесения изменений и несвоевременного приведения продукта в соответствие с требованиями.

Комплект

Это набор файлов, поставляемый пользователю. В процессе создания комплекта процедура установки связывается с устанавливаемыми файлами. Результатом часто является набор сжатых файлов, не представляющих того, что реально будет помещено на систему пользователя. Хорошо бы знать, что применяется в процессе создания комплекта и что получается на выходе. Неплохо разработать и тест, проверяющий наличие нужного числа файлов с приблизительно правильной датой и размером. Для этого могут быть очень полезны приложения, автоматически проверяющие содержимое комплекта.

Сбор всего вместе

Когда у вас есть сборки и процедура установки, следует собрать все вместе в автоматизированный конвейерный процесс. Процесс должен быть создан в самом начале проекта, возможно, это должно быть первой задачей.

Главные шаги цикла сборки таковы:

- сборка образов;
- создание комплекта;
- тестирование комплекта;
- отправка сообщения о прохождении теста или сбое;
- запуск базисных тестов для проверки сборки;
- если тест пройден удачно, копирование в каталог LKGB;
- отправка сообщения о прохождении базисного теста или о сбое.

Ежедневные сборки, комплекты и тесты

Ежедневный процесс сборки, создания комплектов и тестирования задаёт темп реализации проекта. Эти процессы надо запускать еженощно, а чтобы точно понять состояние проекта, — результаты просматривать каждое утро. Только тогда вы сможете принимать грамотные решения о необходимых изменениях. Без этих процессов вы будете действовать вслепую и никогда не узнаете, собирается ли проект воедино (и вообще сможет ли он заработать), пока не будет слишком поздно, и вы ничего не сможете поделать. Всё, что вам останется, — сдвинуть график.

Убеждение

В организациях, где уже приняли эти правила, люди знают, что сборки, установки и базисные тесты могут выполняться ежедневно. Но в других организациях сотрудники могут отнестись к этому скептично. Обычно они слишком заняты, у них нет ресурсов, и они противятся выполнению лишней работы. У кого есть время на эту ерунду? Но выгода огромна, и вы можете внедрить эти принципы. Я рекомендую сначала убедить команду в этих идеях, а затем реализовывать их шаг за шагом: сначала создавать рабочие сборки, затем процедуры установки и, наконец, базисные тесты.

Типичные проблемы

Отсутствие технологов по разработке ПО

Не забудьте учесть в планах и графике вашего проекта мероприятия по технологическому обеспечению. Некоторые команды не учитывают значительный объём работ в этом направлении. В результате их часто ожидают сюрпризы и необходимость корректировки графиков.

Недостаточная автоматизация

Автоматизация сборки, создания комплектов и тестирования очень важна. Выполнение этих задач вручную потребует много времени и усилий, которые могут быть потрачены на что-то другое.

Запоздалая процедура установки

Некоторые команды часто строят исполняемые файлы, но не создают программу установки до самого конца проекта. Такой подход ухудшит видение проекта, а преимущество простого доступа, вытекающее из ранней процедуры установки, теряется всеми членами команды. Столь же часто члены команды обнаруживают, что создание процедуры установки займёт гораздо больше времени и она более сложная, чем думали изначально, а её отладка потребует ещё времени. К сожалению, поздняя разработка чего-либо, в том числе процедуры установки, добавляет риск и способна нарушить график.

Дисциплина

Ежедневное создание сборок, комплектов и тестирование требует много сил. Если дисциплина в команде недостаточно высока или проблемы остаются неразрешёнными, вы не сможете воспользоваться преимуществами, о которых мы говорили ранее. У команды должна быть культура решения проблем. Каждый член команды должен быть дисциплинирован и бороться с проблемами с момента их появления.

Формулирование и планирование проекта.

Требования

В этой главе рассматривается процесс формулирования требований к программному продукту. Каждый член команды разработчиков должен чётко представлять, какую программу нужно создать, для чего она предназначена и каковы её возможности — иначе у вашего проекта не будет ни единого шанса на успех. Проще всего добиться этого понимания с помощью чётко определённого и строго контролируемого набора требований. Но не менее важна возможность улучшения программного продукта и переработки некоторых его фрагментов. Проект должен допускать постепенное улучшение программы вплоть до добавления одних функций и удаления других. Эти две потребности — строгий контроль и свобода развития — часто выглядят взаимоисключающими, поэтому рассмотрим каждую из них.

Для первой требуется чётко сформулированный, подробный и строгий список требований, оговаривающий практически все особенности продукта. Его дополняет жёстко заданный набор процессов, управляющих внесением изменений. Проблема этого метода заключается в трудности создания такого списка требований, особенно при работе в новых, неразработанных областях. Кроме того, он с трудом обеспечивает постепенное улучшение продукта и организацию обратной связи. Даже если создать подробный список требований было бы возможно, то в письменной форме он часто терял бы свою однозначность, а поддерживать его в актуальном виде было бы довольно трудно.

Второй подход утверждает, что достаточно лишь создать простой список требований в общей формулировке. Идея в том, чтобы дать разработчикам свободу принимать решения о реализации основных функций продукта во время его разработки. Более динамичная среда позволит разработчикам оперативно воплощать новые идеи и адекватно реагировать на потребности рынка. Однако этот подход полон неопределённости и риска: трудно планировать рабочий процесс, а управлять — ещё труднее. Это также негативно сказывается на тестировании и создании документации, так как до самого выпуска, т.е. до выяснения истинной картины функциональности продукта, сведений о продукте для начала работы будет недостаточно.

У каждого подхода свои преимущества, но какой же из них выбрать? Нужно, ещё до начала написания кода, установить фундаментальные требования, но при этом иметь возможность вносить контролируемые изменения во время цикла разработки. Давайте обсудим процесс управления требованиями, который позволит их сбалансировать.

Центральная идея проекта

В начале работы над каждым выпуском нужно добиться простого и ясного видения проблемы, при котором задачи и приоритеты проекта стали бы

очевидными для всех его участников; критически важно объединить их усилия и гарантировать, что группа будет работать сообща.

Атрибутом хорошего видения проблемы является центральная идея (лейтмотив проекта), которая сплотит группу и даже всю компанию воедино. Она должна не только направлять усилия при разработке, но и способствовать позиционированию, сбыту и продвижению продукта на рынке. Вокруг неё должны объединиться все группы, обеспечивающие коммерческий успех продукта.

Хотя у крупных проектов может быть несколько таких идей, распыляться всё же не стоит. Основная идея проекта должна быть сформулирована кратко и ясно, в ней должен быть призыв к превосходству в одной-двух областях. Выбрать её нелегко, обычно такая идея является результатом тщательного анализа рынка и состояния бизнеса в данной области. Убедитесь, что достижение цели, поставленной основной идеей, принесёт значительную прибыль.

Мы в NuMega всегда пытались сделать выпуск каждого продукта волнующим событием, претендуя на самый короткий срок его создания, либо на первенство в использовании новых технологий, вплоть до того, что целью ряда наших проектов было получение премий в нашей отрасли. Вот ряд идей, которые в прошлом позволили эффективно объединить наши усилия по разработке:

• закрыть путь на рынок новым конкурентам, предоставив программистам на языке C/C++ продукт с самым полным набором функций по обнаружению ошибок;

• создать продукт для анализа производительности, самый простой в эксплуатации во всей отрасли, и добиться признания этого факта;

• создать самый мощный и функционально насыщенный отладчик ядра Windows NT.

Мы руководствовались этими идеями при выборе функций продукта и в процессе их реализации. Они также играли главную роль, когда приходилось идти на компромисс. Например, если приходилось выбирать одну из двух взаимоисключающих возможностей, достаточно было одного взгляда на центральную идею проекта, чтобы стало ясно, какую из них выбрать.

Поиск и решение пользовательских проблем

Сформулировав центральную идею проекта, надо сосредоточиться на потребностях пользователя. На этом этапе процесса формулирования требований следует рассматривать те проблемы, что необходимо решить пользователю, а не конкретные действия, которые он хотел бы выполнять. Возьмём в качестве примера одну из формулировок идеи проекта из предыдущего раздела. Если нужно предоставить программистам на C/C++ наиболее полный продукт для обнаружения ошибок, то надо выяснить, какие ошибки являются наиболее распространёнными и труднее всего поддаются обнаружению. Вот ещё один пример: если нужен самый простой в использовании продукт для анализа производительности, нужно понять, какие сведения о производительности критичны для пользователей и в каком виде они хотели бы их получить.

Чтобы понимать пользовательские проблемы, важно поддерживать обратную связь с клиентами для проверки сделанных предположений. Лучший

способ убедиться в разумности своих планов и преодолеть внутренние разногласия — иметь обратную связь, заслуживающую доверия.

Из собственного опыта

При работе над BoundsChecker 3.0 было много прений вокруг набора функций продукта. Несколько недель обсуждения этого предмета, порой доходившего до жарких споров, прошли без видимого прогресса. Было принято совместное соглашение оставить этот вопрос, чтобы избежать возобновления споров. Чтобы выйти из тупика и поднять боевой дух группы, мы решили пригласить группу заказчиков и потенциальных пользователей на вечеринку с угощением и раздачей призов. Там мы продемонстрировали разные идеи о возможных функциях программы и попросили приглашённых высказать своё мнение. На основе информации извне стало намного легче прийти к компромиссу и выработать решение, у которого были неплохие шансы на успех.

Формулирование требований

Когда установлено общее видение проекта и достигнуто понимание пользовательских проблем, пора переходить к определению требований. Как сформулировать требования, насколько подробными должны быть формулировки и как ничего не упустить?

Общие и частные требования

Один из лучших способов дать чёткое описание набора требований к проекту — представить его в виде схемы. Самый высокий уровень схемы занимают общие требования. Они объединяют совокупности частных требований, которые, таким образом, можно обсуждать, оценивать, сравнивать и утверждать как единое целое. Нужно иметь возможность анализа общих требований и обладать совершенным пониманием их основных целей. Общих требований не должно быть слишком много, так как каждое в свою очередь генерирует ряд второстепенных требований. Например, в случае компании, которой надо адаптировать имеющееся приложение обработки заказов для работы в Интернете, достаточно пяти общих требований:

• разработать интерфейс на базе браузера;

• повысить производительность до уровня, приемлемого для Web-пользователей;

• организовать рассылку уведомлений о выполнении заказов по электронной почте;

• добавить к программе новые возможности, которые повысят производительность пользователей;

• предусмотреть применение в будущем в качестве клиентской платформы карманных компьютеров.

Каждое общее требование должно подразделяться на несколько частных. С последними могут быть связаны и другие требования, конкретизирующие или

поясняющие функциональность требований более высокого уровня. В результате документация может принять такой вид:

• Общее требование 1
•• Частное требование 1
••• Частное требование нижнего уровня 1.1
••• Частное требование нижнего уровня 1.2
•• Частное требование 2
••• Частное требование нижнего уровня 2.1
••• Частное требование нижнего уровня 2.2

Ниже приводится пример некоторых общих и частных требований, организованных в соответствии с вышеописанной структурой.

• Разработать интерфейс на базе браузера для приложения по обработке заказов.
• Функциональные требования.
•• При размещении заказа:
••• Ввести для каждого заказа следующую информацию (по пунктам).
••• Проверить идентификатор покупателя.
•• Удалить заказ.
•• Проверить статус заказа.
•• Сгенерировать подтверждение заказа.
• Обеспечить поддержку следующих браузеров:
•• Microsoft Internet Explorer версии X.
•• Netscape версии Y.
• Производительность должна быть приемлема для Web-пользователя.
• Требования ко времени реакции системы:
•• Размещение заказа должно занимать менее 3 секунд.
•• Удаление заказа должно занимать менее 6 секунд.
•• Проверка статуса заказа должна занимать менее 4 секунд.
•• Подтверждение заказа должно занимать менее получаса.
• Облегчить использование приложения с помощью новых возможностей:
•• Разрешить заказ нескольких товаров в одном заказе.
•• Разрешить пользователю просмотр его идентификатора покупателя.

Как видно из этого примера, каждое общее требование включает набор поддерживающих его требований, которые конкретизируют или разъясняют содержание «родительского». Каждое поддерживающее требование сформулировано просто и ясно, что позволяет легко проследить его реализацию в данном выпуске ПО. Следует расширять степень детализации спецификации требований, пока не будут описаны все ключевые элементы функциональности и вы не останетесь довольны созданным описанием.

Полнота требований

Определение требований должно быть полным. Рассмотрите все аспекты нового выпуска, даже те, что нельзя свести к набору частных требований. Далее приводится список общих категорий требований, применимый практически ко

всем проектам по созданию программ. Я не предлагаю использовать этот список в том виде, в каком он представлен здесь, хотя возможно и такое; однако при составлении собственного списка требований рассмотрите каждую из следующих категорий.

• Задачи и функции проекта

Каждый участник должен понять ключевые задачи и функции проекта, прежде чем приступать к работе. Эти задачи и функции составляют сущность программного продукта и будут направлять его разработку, а также работу по тестированию и обучению пользователей.

• Пользовательский интерфейс

Хотя при работе над пользовательским интерфейсом придётся дать ответ на два важных вопроса: «Как пользователю выполнить действие X?» и «Как должна выглядеть функция Y?», лучше не пытаться формализовать их, так как это слишком затруднит описание, тестирование и реализацию последовательных улучшений. Вместо этого надо разработать визуальную модель приложения с помощью различных методик конструирования прототипов пользовательского интерфейса. Эта модель и будет спецификацией требований к пользовательскому интерфейсу. (Подробнее об этот см. главу 9. Там же я расскажу об эффективных способах формулирования и анализа требований к пользовательскому интерфейсу программного продукта.) При наличии конкретной платформы, технологий или связанных с бизнесом ограничений, влияющих на структуру интерфейса, важно оговорить их заранее.

• Среда

Необходимо описание программной и аппаратной среды, в которой будет работать продукт. В описании должны быть чётко указаны конкретные версии существующего ПО с учётом новых выпусков, которые могут стать доступными к окончанию работы над проектом. Не забывайте о проблемах, связанных с глобализацией: поддержке ОС, местных языков, валют и различий в часовых поясах.

• Интеграция

Определите потребности, связанные с интеграцией и возможностью взаимодействия с существующими программами и оборудованием. При необходимости интеграции новой программы с существующими решениями, следует указать способ её осуществления и поддерживаемые версии программного и аппаратного обеспечения.

• Производительность

Определите ожидаемую производительность продукта. Обозначьте в простом виде значения параметров, которые нужно достигнуть, а также возможные способы измерения этих параметров. Следует подумать и о времени реакции системы в зависимости от типов нагрузки и потребностей пользователей.

• Установка

Уделите внимание установке ПО. В определении требований должны обсуждаться по крайней мере действия, которые должен выполнить пользователь, чтобы установить ПО, а также действия самой программы установки,

необходимые для завершения процесса установки. Кроме того, укажите платформы, которые должна поддерживать программа установки.

•Тестирование

Требования к тестированию продукта могут не только способствовать существенному повышению продуктивности работы, но и принести дополнительные выгоды. Так, если в программе установки предусмотрен режим, не требующий ручного ввода информации, можно будет автоматически устанавливать и тестировать все ежедневные сборки программы. Не исключено, что программный продукт должен будет поддерживать набор API, позволяющих группе, проводящей испытания, читать любые двоичные файлы, используемые или генерируемые приложением. Это позволит сравнивать файлы, полученные в результате нескольких испытаний программы с последовательно изменёнными параметрами. Также можно заставить программу вести протокол внутренних несогласованностей, который будет полезен при диагностике трудно воспроизводимых сбоев в работе программы.

Детализация требований

Ещё одна проблема, которую придётся решить, — насколько подробно нужно формулировать требования. Разумеется, в данном случае задача в том, чтобы определение было как можно полнее: чем подробнее описано требование, тем легче следить за ходом его реализации. Чем больше аспектов определено заранее, тем больше параллелизма в работе разработчиков и групп, отвечающих за тестирование, обучение пользователей и выпуск программного продукта, так как тогда им проще понять, какой продукт создаётся. Однако часто подробно документировать требования очень трудно и даже невозможно, так как приходится работать в незнакомых областях (так чаще всего и бывает при работе над программными проектами). Как правило, чтобы понять, что именно пытаются создать участники проекта, приходится изрядно поэкспериментировать и испробовать много новых идей. Бывает и так, что поставленная цель оказывается вовсе недостижима. Ниже я опишу способ, позволяющий согласовать потребности в эксперименте и в документировании требований к проекту.

Недостаточно подробное определение обычно является следствием недостаточного понимания. Если недостающие сведения относятся к маркетингу или другим вопросам бизнеса, то разработчики мало чем помогут — это работа менеджера проекта и менеджера по маркетингу. Однако при нехватке сведений о реализуемых функциях, например, когда неясно, как работает та или иная функция, откуда берётся информация или чего хочет пользователь, можно создать прототип пользовательского интерфейса, иллюстрирующий внешний вид этих функций. Если недостающая информация касается технических возможностей, скажем, может ли программный продукт выполнять те или иные действия, можно провести анализ технической осуществимости, а затем создать прототип. Вот как свести воедино информацию из реального мира, экспериментальную работу и творческий процесс в процесс формулирования требований, прежде чем перейти к планированию (рис. 8-1).

Главная идея в том, чтобы заранее выяснить места возможного риска и до начала работы над проектом разработать решения потенциальных проблем.

Анализ осуществимости и прототипы пользовательского интерфейса помогут понять суть проблемы, оценить потребности и снизить общий риск. Эти методики обеспечивают процесс формулирования требований обратной связью с внешним миром и позволяют составлять более детальные планы.

Рис. 8-1. Связь между требованиями, практичностью и созданием прототипа.

О базовых методиках анализа технического риска и создания прототипов пользовательского интерфейса, а также их использование для формулирования оптимальных требований см. главы 9 и 10.

Анализ требований

Когда требования сформулированы, но ещё не утверждены, разумно проанализировать их в целом и каждое по отдельности. Требования к новой программе нужно отбирать очень тщательно. Многие просто берут список требований, не анализируя его с точки зрения коммерческой привлекательности и не удаляя всё, что не вписывается в центральную идею проекта. В итоге получаются программы с плохо организованной функциональностью, не соответствующей назначению программы. Оценке требований и поиску направления, в котором движется разработка вашей программы, должно уделять большое внимание.

«Фрагментация» требований

Отойти от намеченного пути при формулировании требований легко. Так бывает, когда упрямый индивид или целая группа уводит «фокус» требований совсем в другую сторону, чем было задумано. Возможно, было легче формулировать требования для тех функций, определить которые было проще всего. Может оказаться и так, что требования сопряжены с чрезмерным для таких программ риском. Как бы ни было, обязательно должны присутствовать объективный взгляд на требования и оценка их влияния на ход проекта.

Категории требований

Ниже описаны четыре категории требований.

• «Опережающие» и «догоняющие» требования

Первые позволяют продукту обогнать конкурентов по рынку. Они могут описывать просто новое представление данных или возможность поддержки новой платформы. Им не обязательно быть революционными — достаточно, что они дают преимущество в конкуренции на момент выхода программы на рынок. Вторые подтягивают функциональность программы до уровня конкурентов. Они призваны сохранить конкурентоспособность и связаны с решением проблем сбыта и технической поддержки.

Разделение требований на категории «опережающих» и «догоняющих» позволяет проанализировать конкурентоспособность вашей программы. Например, если выяснилось, что в программе реализовано слишком мало опережающих требований или их вообще нет, можно реализовать дополнительные опережающие требования. Стратегия разработки программ в

NuMega всегда включает ряд особенностей, которые делают программу уникальной на рынке и позволяют ей стать или остаться, так сказать, «чемпионом породы».

• Перспективные и ретроспективные требования

Последние направлены на решение проблем, связанных с прошлыми выпусками ПО. Например, для решения проблем с производительностью и удобством использования в последнем выпуске в общем случае служат ретроспективные требования. Так как эти требования — не что иное, как реакция на существующий продукт в существующем окружении, их реализация позволяет улучшить продукт, но не предвосхитить будущие потребности.

Перспективные требования позволяют заранее побеспокоиться о будущих потребностях заказчика. Они основаны на уверенности в том, что заказчик обязательно изменит свои потребности и желания, даже если сам он ещё об этом не знает. Часто перспективные требования базируются на крупномасштабных изменениях в деловой практике (например, на повсеместном внедрении размещения заказов через Web), технологиях (появление платформ, поддерживающих беспроводную связь) или рынка (слияние двух конкурировавших фирм). Перспективные требования труднее всего сформулировать, но, если удастся верно предугадать нужды потребителя и не ошибиться с выбором рынка и функций ПО, результатом будет значительное преимущество перед конкурентами.

Подборка ретроспективных и перспективных требований должна соответствовать потребностям, которые призван удовлетворить продукт, особенностям рынка и задачам выпуска. Скажем, можно быстро (за полгода) подготовить выпуск, в котором будет реализован ряд ретроспективных требований, а также включить в него несколько перспективных требований, чтобы не упустить какую-то интересную возможность.

Перспективные требования вовсе не обязательно являются копией опережающих. Перспективные требования именно предвосхищают потребности, в то время как опережающее требование может быть основано на текущих потребностях клиента, оставленных без внимания конкурентами. Так, введение поддержки диаграмм и графиков и нового одноэтапного процесса ввода данных можно считать опережающими, но никак не перспективными. С другой стороны, поддержку карманных компьютеров можно одновременно рассматривать, как опережающее и как перспективное требование, которое приносит двойную выгоду.

Наглядное представление требований

Чтобы понять сформулированные требования в общем, можно использовать таблицу для анализа требований, расписав их по ячейкам таблицы. Эта таблица имеет вид квадрата. поделённого на четыре равные части. Ниже по одной оси находятся догоняющие и опережающие требования, а по другой — перспективные и ретроспективные требования (рис. 8-2).

Рис. 8-2. Набор требований, представленный в виде таблицы из четырёх ячеек.

Далее приводится описание содержимого каждой ячейки таблицы. Расписав собственные требования по ячейкам такой таблицы, вы поймёте, в каком направлении пойдёт работа.

Ячейка 1.

Вы предвидите будущие потребности потребителей и будете первым производителем, предоставившим соответствующее решение. Эти потребности пока ещё не до конца поняты и не полностью установлены. Вы первопроходец в этой области, поэтому уровень риска довольно высок. Из-за множества «неизвестных» нельзя заранее предоставить подробное определение требований. Основное внимание должно быть уделено созданию и последовательному улучшению прототипа ПО. Потребуется очень быстро пересматривать структуру ПО в процессе разработки, привлечь для тестирования реальных пользователей и обновить требования до начала этапа планирования.

Ячейка 2.

Ряд возможностей ПО, созданного конкурентами, предвосхищают нужды потребителей, поэтому хотелось бы наверстать упущенное. Следует изучить предложения конкурентов, понять, что они сделали правильно, а что — нет, и извлечь урок из допущенных ими ошибок. Риск, связанный с требованиями из этой ячейки, меньше в сравнении с риском в ячейке 1, так как уже существуют программные продукты, способные стать материалом для изучения и извлечения уроков. Однако следует ожидать быстрого изменения рынка и потребностей клиентов, поэтому, прежде чем перейти к формализации требований и созданию плана проекта, придётся затратить много усилий на моделирование технических характеристик и анализ удобства использования.

Ячейка 3.

Наверное, реализация требований из этой ячейки доставит меньше всего хлопот, так как нужно предоставить уникальный в отраслевом масштабе набор возможностей без риска ошибиться в прогнозе тенденций рынка. Поскольку вы работаете с хорошо известным и заслуживающим доверия заказчиком, риск при разработке ПО такого типа обычно связан с верной реализацией функций и своевременным завершением разработки, а не с применением инновационных технологий.

Ячейка 4.

Тактика заключается в расширении функциональности продукта, который уже поставляют конкуренты. Риск в случае такого ПО должен быть относительно невелик, так как вы работаете в основном с хорошо известными функциями и технологиями на устоявшемся рынке. Поскольку риск столь мал, на тестирование удобства использования и последовательное улучшение прототипа уйдёт меньше времени, чем в других случаях.

Помните: задача — обеспечение желаемого коммерческого эффекта при реализации сформулированных вами требований. Хотя вполне можно создать набор требований, распределяющийся по ячейкам таблицы практически поровну, в общем случае это нежелательно, поскольку так можно легко отойти от намеченной цели. Намного лучше сосредоточить большинство требований в одной из ячеек, а остальные требования разместить ещё в одной-двух ячейках.

Определение приоритетов

Определив и проанализировав требования, вы вплотную приблизились к обладанию полным набором функциональности. Но, прежде чем идти дальше, нужно задать приоритеты требований. Это поможет заранее оценить важность каждого требования и понять, как они связаны с другими требованиями.

Почему это так важно

Расстановка приоритетов определяет планирование и распределение задач. Вообще при планировании стараются наметить окончание реализации наиболее критичных требований на максимально ранние сроки. Если сначала сконцентрировать усилия команды на воплощении самых важных требований, можно снизить неопределённый риск, неизбежно присутствующий в любом плане. После завершения реализации критичных функций новая программа уже будет жизнеспособной. Если на этом этапе придётся сжимать сроки или вносить непредвиденные изменения, то вы окажетесь в выигрышном положении и сможете быстро завершить работу над новым выпуском программы, так как её основные функции будут уже готовы.

Коллектив должен заранее прийти к соглашению об уровне приоритета каждого из требований. Лучше, чтобы в момент принятия трудных решений (если такой момент настанет) об исключении той или иной функции вами не владели эмоции и напряжённость ситуации, что может сказаться на принятии решения. Не должно быть никаких сомнений в том, какие функции обязательны, а какие — нет. Приоритетные функции данного выпуска должны быть ясны каждому участнику проекта.

Как это делается

Каждому требованию должен быть назначен тот или иной приоритет. Детализация приоритетов может быть любой, в зависимости от потребностей. Уровни приоритета можно определить следующим образом:

• Необходимые

Обязательно должны быть воплощены в программе, без этого её нельзя выпускать на рынок. Необходимые функции должны быть реализованы и испытаны как можно раньше; этому нужно уделить максимум внимания и все ресурсы.

• Желательные

Присутствие этих требований крайне желательно. При достаточном обосновании от их реализации можно отказаться, но это не уменьшает важности их наличия в программе. Реализация и испытания желательных требований начинается сразу после обязательных.

• Возможные

Эти требования также желательны, но реализуются в последнюю очередь и являются первыми кандидатурами на удаление, если вдруг возникнут проблемы со сроками.

Утверждение требований

Многие ошибочно считают, что сформулировав требования, они готовы к распределению заданий и планированию проекта. Это не так. Нужно выполнить ещё два важных действия: провести техническую экспертизу основных факторов риска, связанных с технологиями, и создать прототип пользовательского интерфейса вашей программы. См. об этом главы 9 и 10.

Представим на минуту, что прототип пользовательского интерфейса уже готов и технологическая экспертиза закончена. Готовы ли вы поставить свою подпись на списке требований? Ещё нет. Нельзя утверждать требования, пока не составлен план работы. Может оказаться, что на исполнение плана должно уйти слишком много времени, и придётся отказаться от реализации части требований. Если требования уже утверждены, то может оказаться, что при их реализации не удастся уложиться в заданные временные рамки. Не стоит утверждать требования поодиночке, пока не будет ясности с другими требованиями.

Управление внесением изменений

Разработка ПО — динамический процесс. Не важно, насколько всеохватывающими будут начальные требования — все равно по мере продвижения по циклу разработки придётся вносить изменения. Постоянно возникают неожиданные проблемы, рождаются идеи, меняются потребности рынка. Однако изменчивость требований — не самая большая проблема. Важнее организовать работу, чтобы поднимать эти проблемы для последующего решения, направлять процесс принятия решений и доводить результаты до сведения коллектива.

Вот ряд фундаментальных принципов, которые нужно учитывать:

• Создавайте команду в составе менеджера проекта и всех руководителей групп, которая будет рассматривать все вносимые изменения

Команда должна регулярно собираться, ответственно подходить к рассмотрению запросов на внесение изменений и гарантировать, что руководитель каждой группы может оценить эффект изменения. Менеджер проекта должен следить, чтобы рассмотрение запросов на внесение изменений шло эффективно, а рассматриваемые запросы не выходили за рамки компетенции команды. Он также должен изучать влияние изменений на план исполнения проекта. При внесении изменений нужно пересмотреть и соответствующим образом изменить план.

• Необходимо не только разрешить, но и стимулировать группы, ответственные за реализацию определённых функций, к улучшению этих функций ПО

Такая свобода поможет им быстрее пересматривать и улучшать продукт. Однако недопустимо, чтобы улучшения частных особенностей или более тонкая настройка негативно отражалась на исполнении проекта. Если изменение влияет на требования или на его внесение уйдёт больше времени, чем допускает план, при обработке запроса на внесение этого изменения нужно обязательно следовать вышеописанной процедуре.

Единственное исключение — обязательные требования. Они являются наиболее важными, и их нельзя изменять как таковые без оценки и санкций извне. От реализации этих функций зависит работа других подразделений компании, и их также необходимо включить в процесс принятия решения. Так, внешнюю экспертизу могут производить менеджеры по продукции, маркетингу, старшие менеджеры и другие ключевые заинтересованные лица.

• Необходимо ознакомить с изменениями всю команду

Эти сведения можно просто разослать по электронной почте или обговорить на очередном рабочем собрании — главное, чтобы в курсе изменений был каждый.

• Все изменения должны быть документированы

Документирование позволяет команде отслеживать и анализировать изменения в проекте. Следует указывать дату внесения изменения, его суть и краткое обоснование. Важно, чтобы на протяжении жизненного цикла выпуска ПО подобная информация хранилась в едином месте, чтобы все участники группы могли обращаться к ней по мере необходимости. Можно вести реестр изменений в начале документа со списком требований или регистрировать в специальном журнале все утверждённые запросы на внесение изменений.

Общие проблемы и решения

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Как изыскать время

Одна из самых распространённых причин для отказа от формулирования требований (а также от затраты усилий на создание прототипов пользовательского интерфейса) в том, что на решение этих задач требуется время. Узнав, сколько времени уходит на это, некоторые спрашивают: а с пользой ли оно тратится? В начале проекта все находятся под давлением потребности поскорее выдать первые результаты, поэтому такой вопрос вполне закономерен.

Ответ однозначен: «Да». Некоторые подходы к разработке ПО подразумевают затрату значительного времени на анализ и вывод подробных формулировок требований. Но подход, рассматриваемый в этой главе (и в следующих двух) предлагает цикл определения требований — он экспериментально проверен и подтверждён положительными отзывами. Это не только не позволяет группе расслабиться, но и даёт возможность проявить творческий подход и поэкспериментировать, прежде чем черновики планов будут созданы и переданы на утверждение. Кроме того, использование прототипов

пользовательского интерфейса и технических решений позволяет увидеть и почувствовать результаты работы над проектом.

Формулируйте сами задачи, а не способы их решения

Спецификация требования должна отвечать на вопрос «что должно быть сделано?», а не «как это сделать?». Ответ на вопрос «как?» проще всего получить с помощью анализа технической осуществимости и создания прототипа пользовательского интерфейса. К сожалению, часто формулировки требований включают описание способов их реализации, что может сузить выбор возможных решений. Вместо этого следует позволить команде задействовать свой творческий потенциал, чтобы генерировать множество возможных решений, а затем экспериментально проверить их в реальном мире.

Рассмотрим в качестве примера вышеупомянутое приложение для обработки заказов. Одно из ключевых требований таково: «Принимая заказ на товар, нужно собрать следующую информацию: X, Y и Z». Заметьте: требование содержит формулировку самой задачи, но не способа её решения. Можно привести пример требования с описанием способа его реализации: «Пользователь должен выбрать в меню пункт New|Create Order и ввести нужную информацию в диалоговом окне». Лучше позволить команде, ответственной за разработку пользовательского интерфейса, самостоятельно найти лучший способ реализации этого требования путём работы с прототипом.

Не упустите главное

В большинстве случаев участники проекта воспринимают формулирование требований в штыки именно из-за плохо налаженного управления требованиями. Проекты страдают из-за «размазанности» и неполноты требований, отсутствия приоритетов и из-за неуправляемых изменений. В этой главе мы сосредоточились лишь на главных вещах, необходимых для поддержания проекта в управляемом и предсказуемом состоянии, не перегружая группу обработкой второстепенной информации и бумажной работой. Не забывайте всё, о чём здесь говорилось, при работе над собственным проектом.

Исследования, оценка технологий и моделирование

В начале любого напряжённого проекта велико искушение принять решения о применении новых технологий, компонентов и платформ лишь на основе общих допущений. Производительность, масштабируемость и даже среду разработки и инструменты нередко оценивают и подбирают «на глазок». Если сделанные допущения верны, считайте себя большим героем, сэкономившим кучу времени. Но в случае ошибки проект с самого начала обречён.

Значит ли это, что любой проект — это лотерея? Да, во многом так оно и есть: не разобравшись в некоторых вопросах планирования, о которых пойдёт речь в этой главе, вы сильно рискуете. Иногда этот риск оправдан, но чаще — нет. Почему? Да потому, что в данном случае все против вас: чем больше вы делаете допущений, тем больше риск.

Не правда ли, здорово было бы знать, как делать предположения с минимальной вероятностью ошибки? И ещё лучше, если бы основные

предположения можно было проверить до утверждения окончательной даты выпуска, правда? В этой главе мы обсудим, как с помощью исследований, оценки технологий и моделирования проверить предположения и не дать проекту сойти с дистанции.

Чем полезны исследования и прототипы

Исследования, оценка и использование прототипов позволят ещё до начала работы над проектом понять все возможности и ограничения технологий, которые планируется применить. Если максимально задействовать эти подходы, то все перечисленное ниже станет намного легче.

• Управление рисками и создание рациональных планов

На ранних стадиях реализации проекта надо определить основные технологические проблемы и наметить пути их решения. Нерешённые технологические проблемы могут внести хаос в реализацию проекта. Фактически это одна из самых распространённых причин срыва планов. Не следует утверждать план или начинать работу над проектом, пока не решены основные технологические проблемы.

• Уверенность в успехе

Когда технологии, подходы и архитектура применяются впервые, мало кто из коллектива верит, что всё заработает. Это и понятно: в отсутствие опыта решения подобных проблем шансы на успех могут казаться призрачными. Однако такого рода сомнения могут стать источником реальных проблем. От недостатка уверенности в успехе проекта страдает не только боевой дух группы, но и производительность труда. Поэтому задача — как можно раньше исключить всякие сомнения и создать уверенность, что проект может быть и будет успешным.

• Прогноз проблем с производительностью

Почти всем ясно, что производительность приложений приобретает всё более важное значение. К сожалению, оптимизация производительности — это не та задача, которую можно отложить на потом, чтобы заняться ею в конце работы над проектом. Здесь нужен другой подход: следует заранее построить модель, воплощающую важнейшие черты архитектуры проекта, и как можно раньше протестировать её, чтобы выяснить, насколько хорошо она масштабируется.

• Прорывы в технологии

Чтобы работать на рынке с жёсткой конкуренцией необходимы исследования. Следует направить часть усилий группы на прогноз нужд потребителей, а также на поиск революционных решений и идей. Прекрасные идеи, новые подходы и хитовые приложения не появляются по волшебству в одночасье — их вынашивают как младенцев.

Исследования

Хоть некоторые и считают исследовательскую работу чисто академическим занятием, она тем не менее играет важную роль в разработке ПО. В этом разделе

мы обсудим основы ведения исследований в приложении к созданию программных продуктов.

Исследования бывают фундаментальные и прикладные.

Первые — это процесс открытий и изобретений в надежде создать что-то полезное. Однако у результата такого исследования может и не быть коммерческого применения. С другой стороны, прикладное исследование на основе логических построений и анализа ситуации в некоторой отрасли ведёт поиск потенциально выгодных решений и пытается превратить гипотезы в конкретные идеи, которые помогут создать некоторый продукт.

Прикладные исследования — наиболее важная форма исследований в контексте этой книги. Они могут обеспечить критически важное преимущество в конкурентной борьбе, особенно на нестабильном рынке, когда потребности пользователей, ПО и аппаратные платформы и технологии претерпевают стремительные изменения. Хотя изменения часто вносят неопределённость, они также дают невероятные возможности группам, способным предвидеть потребности потребителей и задействовать новые технологии для их удовлетворения. Если, занимаясь созданием новой технологии, вы хотите «остаться на плаву» вопреки всем неожиданностям, то параллельно циклу разработки нужно вести непрерывную исследовательскую работу.

Из собственного опыта

Отладчик ядра SoftICE, созданный NuMega, был хитом на рынке программ для 16-разрядных платформ Microsoft DOS и Microsoft Windows. Нам даже без особых проблем удалось заставить его работать в Windows 95. Однако рынок неуклонно двигался к Windows NT, что вынудило нас заняться адаптацией SoftICE для работы с этой ОС, иначе рост прибылей нашей компании неизбежно снизился бы. Но эта задача казалась просто невыполнимой: новая система управлением виртуальной памятью Windows NT делала реализацию многих функций SoftICE чрезвычайно затруднительной и требовала от большинства участников группы разработчиков SoftICE знания недокументированных внутренних механизмов и структур данных Windows. Большинство работников компании (и не только они) сомневалось, что SoftICE когда-либо будет перенесён на Windows NT.

К счастью, среди нас оказался Фрэнк Гроссман, у которого хватило веры в осуществимость этой идеи и желания, чтобы провести соответствующие исследования. Он работал над этой проблемой день и ночь в течение двух недель, пока не создал довольно простой прототип, на примере которого смог продемонстрировать основные методики, необходимые для поддержки Windows NT. Ему достаточно было показать этот прототип группе, и дело было сделано: все поверили, что заставить SoftICE работать в Windows NT всё-таки можно. На реализацию сложных функций программы ушёл почти год, но в итоге мы создали продукт, в появление которого никто не верил. Проведённые исследования позволили нам не только обеспечить стремительный рост потока прибылей, но и

воздвигнуть мощный барьер на пути у конкурентов, а полученные знания мы теперь можем использовать при разработке других продуктов.

Как это делается

Ниже описаны разные модели нахождения оптимального баланса между исследованиями и разработкой. Первая требует наименьшей затраты ресурсов, последняя — наибольшей. Если ваша компания невелика или просто не хватает ресурсов. можно начать с первой модели, по ходу дела она. может перерасти в другие.

• Проведение исследований во время работы над неосновными выпусками программы

Программные продукты развиваются циклически: сначала выходит основной выпуск, затем появляется ряд неосновных. Выход неосновного выпуска можно охарактеризовать как тактический шаг, т.е. в неосновных выпусках реализованы дополнительные функции и усовершенствования, улучшающие и расширяющие возможности продукта, уже присутствующего на рынке. Неосновные выпуски обычно появляются быстро, а их создание сопровождается меньшим риском, нежели создание основного выпуска. В силу своих особенностей неосновной выпуск — замечательный способ дать некоторым из ведущих разработчиков передышку от рутинных задач, во время которой у них есть шанс заняться исследовательской работой. Общее требование: к концу работы над неосновным выпуском исследование надо закончить.

Помимо самой возможности проведения исследований, совмещение исследований и работы над неосновными выпусками даёт участникам группы целый ряд других преимуществ. Ведущие разработчики могут перевести дух и заняться другими проблемами. Это реальная возможность дать главным «дарованиям» расслабиться, чтобы они не «перегорели», а другим членам коллектива — возможность выйти в лидеры. Взаимное обучение и возможность роста талантливых участников критичны для устранения текучести кадров и непрерывного роста мастерства группы.

• Дайте проявить себя каждому ведущему разработчику

Если вам посчастливилось иметь в группе двух и более способных ведущих программистов, пусть они сменяют друг друга на посту ведущего разработчика от выпуска к выпуску. В то время как один из них работает над новым выпуском, остальные могут заняться исследованием новых технологий и поиском новых идей. Разработчик, до этого занимавшийся исследованиями, несёт в группу энтузиазм и знания, накопленные за время исследований. Это пробуждает у других участников группы желание продолжить исследовательскую работу.

• Кандидатуры ведущих исследователей.

Если сложность, размеры и число продуктов растут, то весьма вероятно, что вскоре понадобится провести ряд исследований. При этом надо подумать о найме исследователей или о переводе на исследовательскую работу некоторых специалистов высокого уровня. У последних, помимо профессиональной этики и способности к независимому мышлению и действию, должны быть развитые навыки общения. Таких людей непременно нужно включить в команду,

разрабатывающую продукт, хотя не обязательно, чтоб они принимали участие в её работе ежедневно. Такое разделение обязанностей может дать замечательные результаты, но может стать и причиной отчуждения между командой разработчиков и исследователями. Ответственность за то, чтобы этого не произошло, целиком и полностью лежит на ведущих исследователях и менеджере проекта. Регулярное общение, как формальное, так и неформальное, снижает вероятность возникновения барьеров между группами.

Остаётся обсудить, на чём сосредоточить усилия исследователей. Если работа идёт в среде с небольшими ресурсами, позаботьтесь о том, чтобы шансы на успех исследовательской работы были максимальны. Следует разумно распределить усилия исследователей. В общем случае приоритетными являются следующие направления:

• Анализ тенденций и перспектив рынка

Каждые 3-4 года на рынке появляются новые, более совершенные технологии. Независимо от того, связаны ли новшества с графическим интерфейсом пользователя, клиент-серверными продуктами, моделями компонентных объектов или Интернетом, всегда следует идти в ногу с фундаментальными нововведениями и стараться, чтобы большие перемены не оставили вас позади. Направляйте исследования на поиск, анализ и мониторинг серьёзных изменений на рынке. Однако не заходите в своих исканиях так далеко, чтобы потерять связь с реальностью.

• Отбор новых идей

Новые идеи, связанные как с продуктами, так и с технологиями, могут приходить в любое время и из любых источников — и внутренних, и внешних. Одни идеи могут стоить миллионы, в то время как другие могут быть лишь напрасной тратой времени. У исследователей должен быть навык быстрого отбора хороших идей и отсеивания плохих.

• Анализ инноваций и направлений работы конкурентов

Одна из самых важных областей исследования — анализ инноваций и направлений работы конкурентов. Чтобы успешно состязаться с ними, надо понимать их технологии, знать их сильные и слабые стороны.

По завершении исследовательского проекта или при смене приоритетов исследования важно задокументировать результаты и сделать их доступными коллективу. Необходимо довести до сведения группы все без исключения результаты исследований, как положительные, так и отрицательные. Если исследовательский проект действительно обладает недюжинным потенциалом, было бы разумно создать прототип и продемонстрировать группе его возможности, пояснив принцип работы.

Из собственного опыта

К концу работы над BoundsChecker 3.0 энтузиазм нашего ведущего разработчика изрядно поубавился: проект отнял много времени и сил, и необходимость перемен была очевидна каждому его участнику. В это время возникла ещё одна задача: нужно было разобраться, что может дать новая технология от Microsoft под названием COM для наших продуктов. COM

привлекла большое внимание, её даже считали «дорогой в будущее». Однако мы смутно представляли себе её суть, поэтому решено было взяться за обе проблемы одновременно.

В то время как остальная часть группы работала над следующим (неосновным) выпуском продукта, ведущий разработчик уделял все своё время изучению внутренней организации COM и моделированию новых функций BoundsChecker. Спустя три месяца, когда новая версия была практически завершена, главный разработчик был готов присоединиться к работе над BoundsChecker 4.0. Ему удалось не только восстановить свои силы, но и обучить остальных участников группы принципам работы с COM и показать им рабочие прототипы новых функций. Так что для начала наше положение было весьма неплохим: вовсю шли поставки BoundsChecker 3.0, только что закончена разработка версии 3.1, а мы уже обладали фундаментальной технологией для следующего выпуска программы.

Оценка технологий

Прежде чем приступать к проекту, обязательно нужно разобраться в технологии, намеченной для использования в нём. Это особенно важно для проектов, в реализации которых будут задействованы новые инструменты, компоненты, платформы или решении.

Сегодня практически ни одна программа не создаётся на основе одной технологии. Например, в типичном современном Web-приложении используются функции ОС, графические библиотеки, компоненты от сторонних разработчиков, Web-серверы, серверы транзакций и баз данных, поэтому приходится разбираться в возможностях самых разных технологий. Следует ещё до начала работы над проектом выяснить, соответствуют ли возможности каждой намеченной для применения новой технологии нуждам проекта. Оценка технологий позволяет решать поставленные вопросы путём тестирования и совместного обсуждения, что позволяет обнаружить новые проблемы, о существовании которых никто даже не подозревал до начала использования этой технологии.

Просто удивительно, как часто разработчики считают, что для использования новых технологий достаточно одних предположений или сведений, полученных из прессы или телеконференций. Любая команда должна тщательно изучить все новые технологии, которые она собирается применить в работе над проектом. Если значительную долю задействованных в проекте технологий составляют новые (для рынка или для самой команды), до начала работы над проектом потребуется затратить некоторое время на изучение новых технологий.

Как это делается

Оценивая технологию, нужно дать ответы на следующие вопросы.

• Возможности технологии:

обладает ли новая технология возможностями, необходимыми для реализации проекта?

• Качество: приемлем ли уровень качества технологии?

• Совершенство: обеспечивает ли технология должную производительность, масштабируемость и устойчивость?

• Поддержка: обеспечена ли новая технология адекватной поддержкой?

• Простота использования: не слишком ли сложна новая технология в использовании и при отладке?

• Профессионализм команды: хватит ли у команды мастерства для применения этой технологии?

На самом деле собственно процесс оценки технологии не столь сложен, но нужно провести его довольно быстро, поскольку помимо всего прочего, именно результаты оценки технологий определяют срок утверждения окончательного плана проекта. Даже если время для вас — роскошь, при оценке любой технологии не забывайте:

• формулировать критерии: определяйте свои потребности заранее и делайте это как можно точнее с помощью критериев, данных выше;

• использовать сформулированные критерии при оценке: объективно оценивайте результаты, опираясь на факты, а не на мнения;

• учитывать отзывы заказчиков: собирайте отзывы заказчиков (как положительные, так и отрицательные) о результатах оценки; не забывайте интересоваться мнением заказчиков: удовлетворят ли новые технологии их нужды.

Моделирование

В начале работы над проектом почти всегда возникает ряд важных вопросов, связанных с реализацией той или иной технологии. Моделирование — важная методика, которая поможет получить необходимые ответы.

О чём пойдёт речь

Создание прототипа — важный этап, который любая группа разработчиков может осуществить ещё до начала работы над проектом. Работа с прототипом поможет понять, как эффективно воплотить ключевые функции программы, оценить сложность реализации ключевых технологий и необходимое для этого время, а также свести к минимуму общий риск ошибок и срыва планов.

Рассмотрим примеры того, что может случиться, если отказаться от работы с прототипом, Определив все компоненты программы, команда решила сначала реализовать её инфраструктуру, так как без этого компонента, обычно самого важного и самого сложного, ничего не работает. В результате этот компонент был спроектирован и построен без предварительной работы с прототипом. Когда он был готов, группа решила подключить к нему другие. Но после интеграции новых компонентов стало ясно, что возможности инфраструктуры недостаточны, конструкция её плоха или не масштабируется. После этого приходится искать места, где и что пошло не так, проектировать и кодировать все заново. Перепроектирование программы и изменение её кода во время цикла разработки, очевидно, приведёт к задержке выпуска ПО.

Рассмотрим ещё один пример, на сей раз с противоположным сценарием. Участники группы отдают себе отчёт в том, что нельзя строить компоненты инфраструктуры проекта, не поняв технических требований других частей системы, поэтому решено сначала создать полную спецификацию системы. Но эта задача оказалась затруднительной, так как не все проблемы, с которыми придётся столкнуться, известны заранее. Фактически здесь возникают сплошные вопросы, на которые никто не знает ответа. Конечно, можно попытаться действовать наугад в надежде, что всё будет хорошо, но это слишком рискованно.

Все эти проблемы позволяет решить прототип. В первом примере работа с прототипом помогла бы заранее смоделировать систему. Это позволило бы понять, как собрать все компоненты. Во втором примере работа с прототипом подсказала бы проектные решения.

Из собственного опыта

Оба рассмотренных выше примера взяты из работы над реальными проектами. На заре нашей деятельности мы не уделяли должного внимания созданию прототипов тех элементов, использование которых таило в себе наибольший риск или неопределённость. Оглядываясь в прошлое, понимаешь, что всех проблем удалось бы избежать, если бы конструкция программы была заранее проверена с помощью прототипа.

Как это делается

Прототип — не законченная программа, но он даёт возможность получить фактические данные, которые позволят принимать более удачные решения. Важнее всего — время, поэтому нужно действовать быстро. Ниже описаны этапы кратчайшего пути, который, однако, позволяет создать вполне приличный прототип.

• Определите ключевые факторы риска

Первый этап — создание списка основных вопросов, на которые нужно ответить. Изучите все вопросы, в которых нужно разобраться, чтобы разработать точный план. Если вопросов много, нужно определить приоритетные и проработать их в первую очередь. Помните: нужно сосредоточиться только на ключевых моментах, а не на всех неизвестных.

• Составьте план экспериментов

Следует создать план экспериментов, которые помогут ответить на поставленные вопросы. Можно искать ответ на каждый вопрос посредством отдельного эксперимента или решать несколько проблем одновременно, проводя ряд экспериментов.

Независимо от числа экспериментов нужно смоделировать взаимодействие любых ключевых технологий или компонентов. Здесь задача заключается в том, чтобы проработать продукт «вширь», а не «вглубь», т.е. охватить максимально возможное число функций, а не пытаться полностью воспроизвести какую-то одну из них. Вы не поверите, как часто мне приходилось встречать проекты, в которых возникали катастрофические проблемы при попытке собрать воедино все

фрагменты программы. Обычно причина была в том, что в течение первых недель реализации проекта этим проблемам не было уделено внимания.

• Попробуйте сымитировать конечный результат

Одно из ключевых требований, выполнив которое, можно считать создание прототипа завершённым, — имитация конечного результата. Для этого придётся заглянуть в будущее, чтобы увидеть программу в окончательном виде и попытаться заранее смоделировать её ключевые составляющие, Чтобы преуспеть в этом, придётся ограничиться созданием модели на основе набора временных компонентов и API, имитирующих готовую программу. Возможно, часть функций придётся запрограммировать жёстко, для других вообще написать заглушки, а реальные данные заменить имитационными. Все это допустимо на данном этапе — ведь создаётся всего лишь эмулятор реальной программы. Главная задача сейчас — создать «скелет» программы, а «мясом» он обрастёт позже.

• Используйте существующие наработки

Ещё один важный способ ускорения создания прототипа — использование существующих решений. Вовсе не обязательно все писать «с нуля». Некоторые наиболее успешные прототипы появились в результате модификация копии исходного текста рабочей программы.

• Оценивайте результаты

Когда прототип готов, не забудьте оценить результаты своей работы. В частности, работая с прототипом, можно оценить производительность на макроуровне, необходимый объём памяти и то, как она используется. Можно определить и сложность внедрения, и качество технологии, а также попытаться разобраться в её принципах. Короче говоря, какими бы ни были потребности, нужно выжать из прототипа максимум пользы.

• Документируйте результаты.

Это полезно не только для сегодняшних участников группы, но и для будущих. Если при работе с прототипом обнаружились серьёзные проблемы, то они скорее всего снова возникнут и в будущем. Каждый участник группы должен знать, почему принято то или иное важное решение. Со временем у вас соберётся целая библиотека проектных заметок, которая станет историческим документом проекта.

Из собственного опыта

Во время работы над BoundsChecker 5.0 разработчикам NuMega пришлось почти полностью переписать внутренние компоненты программы. При этом работа шла в основном на двух фронтах: обновление систем сбора и анализа информации. Из-за сложности проекта мы испытывали большой соблазн сначала довести до конца конструирование системы сбора данных, а затем закончить систему анализа. Но опять же в силу сложности проекта мы пришли к выводу, что лучше создать прототипы для обеих систем, чем тратить время на создание подробных спецификаций. Было решено смоделировать сбор части нужных данных и написать лишь части кода для анализа только этих данных. Если программа функционировала нормально, выводилось простое диалоговое окно с сообщением, что все работает.

Спустя неделю один из программистов зазвал меня в свой кабинет и продемонстрировал маленькое простенькое диалоговое окно. Прототип работал! Теперь мы знали, что все задуманное осуществимо от начала до конца и серьёзных проблем с производительностью не предвидится. Следующие две недели мы по очереди наращивали все функции, обретая все большую уверенность в успехе. Таким образом, окончательная архитектура и конструкция программы были существенно улучшены. Через три недели у нас был готовый проект, который мы могли точно спланировать. В конечном итоге это позволило нам сэкономить кучу времени при его реализации. Я не говорю, что после всё было прекрасно, но без этих простых действий у нас бы не хватило уверенности, знаний и понимания, чтобы правильно спланировать проект.

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Не торопитесь

Как уже не раз было сказано, разработчики часто пытаются работать с новыми технологиями, основываясь лишь на допущениях. Эти допущения превращают проект скорее в азартную игру, чем в серьёзную техническую работу. Не торопите события: хотя некоторые проблемы можно и должно решать с ходу, всё же следует определить ключевые потребности и убедиться, что новые технологии в состоянии удовлетворить их. В этом случае вы сможете предвидеть проблемы и лучше подготовитесь к их решению, а также отреагировать на возникшие проблемы на более ранних этапах цикла разработки проекта.

Не увлекайтесь моделированием отдельных функций

Часто возникает искушение создать прототип какого-либо компонента без учёта контекста, не принимая во внимание характер его применения. Хотя сосредоточиться на узкой задаче много проще, в этом таится большая опасность. Моделирование не должно концентрироваться на отработке какого-то одного компонента, важно отработать совместную работу всех критически важных компонентов. Я очень рекомендую создавать общесистемные прототипы, в которых собраны воедино все критически важные фрагменты системы, даже если при этом приходится использовать искусственные данные, жёстко прошивать вызовы API или подменять их заглушками. Тем не менее в этом случае удастся составить хорошее представление о проблемах с интеграцией компонентов и убедиться в проектных решений на уровне системы.

Не оставляйте анализ производительности напоследок

Большинство считает, что производительность — это что-то, что «добавляется» к программе в конце цикла работы над проектом. Хотя настройка разного рода параметров, несомненно, может и должна выполняться после сборки всех частей проекта, также верно и то, что на этом этапе объём возможных изменений весьма ограничен. Неуместно вносить фундаментальные изменения в

архитектуру или внутреннюю структуру продукта в последние недели работы над проектом.

Необходимо заранее проанализировать производительность на макроуровне: сначала на прототипе, а затем после сборки всех основных фрагментов программы.

Пользовательский интерфейс

Думаю, ни один участник команды не станет спорить с тем, что хороший пользовательский интерфейс — ключевое условие успеха продукта. Увы, на этом согласие заканчивается. В командах, одолеваемых проблемами с пользовательским интерфейсом, нередко отсутствует работа с прототипом пользовательского интерфейса, или члены команды не могут договориться о том, как организовать эту работу. Часто проблемы с пользовательским интерфейсом — самое серьёзное испытание, грозящее сорвать план реализации проекта.

Если в этом плане ваша команда не отличается от других, то скорее всего самые горячие споры разгораются при обсуждении вопросов, касающихся пользовательского интерфейса: что и как интерфейс должен делать и на что должен быть похож. Порой эти споры длятся в течение всего времени работы над проектом и становятся причиной столкновений участников команды, задержек в работе и фальстартов. Хуже того, часто проблемы с интерфейсом, о существовании которых никто не догадывался, неожиданно всплывают на поздних стадиях реализации проекта. Чтобы что-то изменить на этом этапе, разработчикам, тестировщикам и техническим писателям приходится многое переделывать, так что в этом случае дело пахнет серьёзным срывом планов.

Решение этой проблемы заключается в создании прототипа пользовательского интерфейса на ранних стадиях цикла разработки. Прототип следует как можно быстрее тестировать, оценивать и улучшать, пока не будут решены основные проблемы. В этой главе мы обсудим значение прототипа пользовательского интерфейса, а также поговорим о способах создания прототипов при работе над самыми разными проектами. Я не буду касаться здесь всех составляющих хорошего интерфейса, так как они могут существенно варьировать в зависимости от платформы. Вместо этого мы рассмотрим, как организовать в цикле разработки ПО эффективную работу по созданию пользовательского интерфейса, которая, однако, не потребует мощного оборудования, специальных ресурсов и существенных затрат времени. В завершение мы обсудим роль специалиста по инженерной психологии и рассмотрим роль этой ключевой фигуры в руководстве работой по созданию пользовательского интерфейса.

Прототип пользовательского интерфейса

Прототип — это наглядная модель пользовательского интерфейса. В сущности это функционирующий «черновик» интерфейса, созданный на основе ваших представлений о потребностях пользователей. Прототип может принимать

множество различных форм, от бумажных макетов до реальных программ, имитирующих работу пользовательского интерфейса (способы создания прототипов мы обсудим ниже). Однако независимо от формы прототип должен давать команде чёткое представление о способе взаимодействия пользователя с программой.

Почему прототип необходим?

Как сказано в главе 8, требования к продукту определяют основные задачи, которые приходится выполнять пользователям. Однако в описании требований не сказано, как пользователи будут это делать. Ответ даёт прототип пользовательского интерфейса. Им занимается команда разработчиков программы и в большей степени — специалист по инженерной психологии. После создания прототипа появляется возможность привлечь пользователей для проверки деталей выбранного вами решения. Результаты тестирования позволяют убедиться, что пользовательские потребности были поняты правильно. Важнее всего при этом возможность легко вносить изменения и пробовать новые идеи, руководствуясь результатами тестов.

Прототип даёт команде разработчиков множество преимуществ.

• Возможность сосредоточиться на решении ключевых задач.

Необходимо сосредоточить усилия команды на самых важных частях продукта. Важнейшие части продукта — это ключевые задачи, наиболее ценные, нужные и важные для пользователя. Все участники команды (разработчики, тестировщики, технические писатели и др.) должны чётко представлять себе суть ключевых задач и не жалеть усилий для их правильного понимания. Отчасти верно, что остальная функциональность просто не имеет значения, если трудно решать основные задачи или не ясно, как это делать.

Из собственного опыта

Когда в NuMega решили создать TrueTime, новую программу-профайлер, было ясно, что интерфейс для неё придётся создавать «с нуля». Но на что должен быть похож новый интерфейс? Какие действия наиболее важны для пользователя? Ознакомившись с положением в отрасли, мы обнаружили массу хороших продуктов, в целом составляющих весьма развитый набор ПО. Однако все существующие программы перегружали пользователя информацией, что, с нашей точки зрения, является недостатком, так как это очень затрудняет выполнение самой важной задачи продукта — поиск проблем с производительностью программ. Теперь мы знали, к чему стремиться: нужно сделать так, чтобы поиск проблем с производительностью с помощью нашей программы требовал не больше трёх щелчков. Вся команда сообща работала над созданием интерфейса, чтобы решить эту просто сформулированную, но такую трудную задачу. Мы решили не усложнять интерфейс и разработали новый подход к навигации по сложной иерархии функций, в которой зачастую таятся проблемы с производительностью. В результате первый выпуск TrueTime принёс нам премию «Best of Show», присуждаемую журналом Comdex/Byte Magazine. Основным фактором успеха была концентрация усилий команды на решении определённых задач.

• Точность

Одна из самых серьёзных проблем при разработке программы — переделки по причине многократных изменений конструкции и реализации пользовательского интерфейса. Чтобы уложиться в сроки, нужно исключить крупные переделки. Хотя получить совершенный продукт с первого раза практически невозможно, надо как можно раньше скомпоновать основные элементы пользовательского интерфейса. Мелкие изменения вполне допустимы, но большие изменения могут привести к катастрофическим последствиям, негативно отражаясь на времени разработки, качестве продукта и документации.

• Планирование

Нельзя составить точный план, не зная структуры пользовательского интерфейса. Если интерфейс будет изменяться по ходу цикла разработки, то в план проекта также придётся вносить соответствующие изменения. Кроме того, необходимо предохранять план от перенапряжения, которое возникает, когда работа тестировщиков и технических писателей сдерживается изменениями пользовательского интерфейса. Таким образом, перенапряжение плана ведёт к задержке работы над проектом в целом. Помните: следует вести работу над всеми частями проекта в параллели с разработкой программы. Планирование требует чёткого понимания структуры пользовательского интерфейса уже в начале работы над проектом.

• Документация

Вам обязательно понадобится документация, описывающая работу с программой. Очевидно, что документация тесно связана с интерфейсом программы. Команда технических писателей должна быть в курсе проблем, с которыми предстоит столкнуться пользователям, а также знать, как их решить с помощью программы. Если интерфейс не довести до ума в начале работы над проектом, техническим писателям придётся постоянно нагонять разработчиков, что сделает их параллельную работу невозможной.

• Тестирование

Тестирование программы также сильно зависит от её интерфейса. Традиционно тестировщики просят разработчиков предоставить им подробные спецификации функций программы, точно описывающие все особенности её работы. К сожалению, полностью описать пользовательский интерфейс, особенно современный, практически невозможно. Лучшая замена описанию — прототип. Если предоставить тестировщикам прототип интерфейса, то, ознакомившись с программой и разобравшись в основах и принципах работы её функций, предназначенных для конечных пользователей, они смогут улучшить разработанные ими планы тестирования. Это не значит, что описание функций не имеет значения или можно пренебречь им, просто я хочу подчеркнуть, что прототип — один из лучших способов демонстрации особенностей продукта. Обладая прототипом пользовательского интерфейса, тестировщики смогут изучить продукт вдоль и поперёк, а разобравшись в работе функций программы, команда сможет подготовить лучшие планы тестов и испытаний до окончания разработки программы.

Создание прототипа

Возможно, дочитав до этого места, вы спросите: «Все это здорово, но как создать прототип?» Подход к созданию прототипов пользовательского интерфейса, который исповедует NuMega, основан на трёх простых принципах. Сначала мы определяем наиболее важные задачи, которые приходится решать пользователям. Затем мы быстро моделируем эти задачи на ранних стадиях работы над проектом, ещё до подготовки плана проекта в окончательном виде: это увеличивает шансы правильно оценить объём предстоящей работы. Наконец, мы очень быстро доводим прототип, внося в него ряд последовательных изменений, демонстрируя как внутренние, так и внешние его особенности. Такой подход позволяет тестировать прототип вместе с пользователями. Возможность быстрой доработки прототипа — ключевой фактор успеха, позволяющий как можно раньше привести его к окончательному виду.

Решив эти задачи, команда получит прототип пользовательского интерфейса, понятный всем её участникам ещё до создания самой программы. Только подумайте, какое значение это имеет для проекта: разработчики, тестировщики, технические писатели — короче, все получат чёткое представление о внешнем виде программы даже раньше, чем она будет создана. Представьте, насколько возрастёт эффективность и производительность труда каждого участника команды благодаря прототипу пользовательского интерфейса. Кроме того, администраторы, менеджеры по продукции, работники из отделов сбыта и технической поддержки смогут «увидеть» программу раньше, чем она появится на свет. Это поможет устранить равнодушное отношение к проекту, создать уверенность в его успехе и предвидеть возможные проблемы — в общем, создать особую атмосферу работы с высокими технологиями, направляющую усилия всех участников проекта в единое русло. И не будем забывать о самом важном: чем раньше будет протестирован интерфейс, тем больше шансов на то, что получится хороший продукт, так как тогда множество людей смогут познакомиться с программой и опробовать её прежде, чем она будет написана.

Хотя описанный сценарий очень похож на идеал, его можно реализовать при наличии соответствующих усилий и навыков. Давайте познакомимся с каждым из трёх этапов этого сценария поближе.

Определение ключевых задач

На первом этапе создания пользовательского интерфейса нужно определить самые важные задачи, которые потребуется решать пользователям. Число задач может варьироваться в зависимости от сложности продукта; попробуйте выделить хотя бы следующие категории:

• Задачи, которые скорее всего придётся решать новым пользователям программы

Здесь надо понять потребности новичков и сделать так, чтобы они как можно скорее преуспели в решении своих проблем с помощью вашей программы. Программа должна вызывать у пользователей не ощущение беспомощности, а стимулировать их к дальнейшей работе с ней. Если известен набор вероятных действий пользователя, то можно оптимизировать интерфейс под их потребности.

• Задачи, которые чаще всего решают постоянные пользователи программы

Нужно постараться не разочаровать постоянных пользователей, безошибочно определив их ключевые задачи. Успех здесь даёт замечательный шанс удовлетворить потребности пользователей на долгое время. Соответственно следует сосредоточить основное внимание команды разработчиков на этих задачах, которые нужно максимально обогатить полезными возможностями, качественно реализовать и хорошо описать в документации.

Виды прототипов

Когда основные задачи определены, всё готово к созданию прототипа пользовательского интерфейса. При этом очень важно выбрать инструмент, позволяющий создать прототип легко и быстро. Затем надо проверить свои варианты дизайна прототипа, разрешить все проблемы и вновь оперативно проверить результат. Познакомимся с наиболее популярными методиками создания прототипов и посмотрим, как с их помощью решать собственные проблемы. Вот эти методики по порядку, начиная с самой лучшей.

• Прототипы на бумаге

Для создания такого прототипа нужно просто нарисовать фрагменты пользовательского интерфейса на бумаге. Чтобы облегчить художникам и пользователям работу с таким прототипом, рисовать каждый элемент интерфейса надо на отдельном листе бумаги соответствующего размера. Расположите нарисованные меню, панели инструментов, командные кнопки, поля и другие элементы так, чтобы результат напоминал пользовательский интерфейс программы. Например, каждый раскрывающийся список и каждое диалоговое окно в этом случае будет на своём кусочке бумаги. Не обязательно добиваться точности воспроизведения — достаточно, чтобы эти кусочки бумаги давали чёткое представление об элементах пользовательского интерфейса.

Преимущество бумажных прототипов в том, что их легко собирать и изменять. Изолируя основные элементы интерфейса, можно легко изменить ход управления, условия работы, а также их расположение или размер. Нетрудно стирать линии или даже перерисовывать отдельные страницы или прототип целиком. Благодаря тому, что бумажные прототипы так легко менять, их можно обновлять и модифицировать прямо во время демонстрации конечным пользователям, это позволяет тут же проверять новые идеи.

Кроме того, бумажный прототип не страдает от проблем, связанных с программированием, установкой и других помех, неизбежных при разработке ПО. Это его преимущество, так как лучше тратить время на тестирование и доводку пользовательского интерфейса, а не на решение технических или механических проблем, связанных с его созданием.

С появлением более совершенных графических программ и инструментов для разработки стало проще создавать реальные изображения элементов интерфейса и распечатывать их на бумаге, чем рисовать их от руки. Следует выбирать тот или иной метод на основе личных предпочтений, поскольку важные преимущества есть у каждого из них: оба позволяют легко создавать и быстро доводить прототипы.

Из собственного опыта

Бумажные прототипы могут быть чрезвычайно эффективны, но если команда не верит в результативность этой методики, использовать её нельзя. Чтобы побороть эту проблему в NuMega, мы отправили всю команду на однодневный методический курс, который читали в институте UEI (Usability Engineering Institute). Особое внимание в этом курсе уделялось работе с бумажными прототипами. Это был замечательный способ продемонстрировать участникам команды, насколько важны и эффективны могут быть прототипы на бумаге. Этот курс изменил наши представления о разработке ПО.

• Инструменты RAD

Создание прототипов с помощью инструментов для быстрой разработки приложений (RAD, Rapid Application Development), — вероятно, самая популярная методика. Подходящим можно считать любой инструмент, позволяющий быстро создать наглядную функционирующую модель пользовательского интерфейса.

Одно из преимуществ инструментов RAD в том, что они позволяют создавать прототипы очень быстро (хотя бумажный прототип, как правило, делается быстрее). Ещё одно преимущество — реализм полученных прототипов. Многие думают, что прототипы, созданные с помощью RAD, обеспечивают более эффективное тестирование, чем бумажные, поскольку в первом случае тестированию подвергается настоящая программа. Однако программисты часто застревают на кодировании таких прототипов и напрасно теряют драгоценное время. Они пытаются довести прототип, улучшая его, а не реальный интерфейс. Другая проблема в сильном искушении перенести код прототипа прямо в рабочую программу, невзирая на его недостаточную проработанность и несовершенство дизайна.

• Описания

Создание прототипов пользовательского интерфейса с помощью описаний, вероятно, вторая по популярности методика, однако наименее ценная из трёх представленных здесь. Описания пользовательского интерфейса страдают от трёх недостатков. Во-первых, их интерпретация часто неопределенна. Маловероятно, что все участники команды смогут одинаково воспринять и понять даже подробное описание. Во-вторых, описание трудно протестировать и оценить. Вряд ли вы рискнёте кинуть 20-страничное описание интерфейса на стол пользователю с просьбой прочитать его. Применять описания в качестве эталонов также очень трудно. Наконец, даже при наличии отзывов, поддерживать точность и внутреннюю согласованность описаний зачастую нелегко.

Повторная оценка и доводка

Получив прототип, можно приступать к его проверке с помощью реальных пользователей. Для этого нужно попросить пользователей выполнить определённые вами ключевые задачи с помощью только что созданного прототипа. При этом нужно выяснить, что работает хорошо, а что плохо, и внести в прототип соответствующие изменения, чтобы решить обнаруженные проблемы.

Рассмотрим пример с бумажным прототипом. Если вы попросите пользователя выполнить некоторую задачу, ему придётся «щёлкнуть» ряд элементов интерфейса. При этом часть работы придётся делать вам, управляя

механикой интерфейса и имитируя для пользователя работу компьютера. Вам придётся собственноручно выкладывать перед пользователем новые «диалоговые окна», наблюдать затем, какие «кнопки» он выбирает, и убирать «окна», когда пользователь «щёлкает ОК»

Сколько времени занимает доводка интерфейса? Обычно немало. Приходится до 20 раз менять структуру интерфейса в зависимости от приложения. Следует вносить столько изменений, сколько потребуется, и доводить дизайн, пока не будет достигнут значительный прогресс. Помните: идея в том, чтобы быстро проработать множество вариантов дизайна, пока форма прототипа не станет более-менее постоянной. Полное представление о прогрессе прототипа дают результаты тестов. Удалось ли облегчить работу пользователей? Уменьшилось или увеличилось среднее время, которое пользователь тратит на решение некоторой задачи? Смогла ли последняя партия пользователей легко и быстро выполнить свои задачи или они столкнулись с проблемами? Ответы на эти вопросы позволят выяснить, как обстоят дела в работе над программой и сколько ещё предстоит сделать.

Из собственного опыта

Во время разработки TrueCoverage, первого продукта NuMega для отображения результатов исполнения кода, команде снова пришлось искать форму пользовательского интерфейса. Чтобы помочь нам в этом, наш специалист по инженерной психологии (совмещавший эту должность с работой технического писателя) вдвоём с ведущим разработчиком создали бумажный прототип пользовательского интерфейса. Затем мы попросили других участников команды выполнить ряд задач, самых важных, по нашим оценкам. Когда новоявленные пользователи закончили свою работу, мы проанализировали их успехи и неудачи. Мы также собрали их отзывы и опробовали ряд новых подходов, решая проблемы, с которыми они столкнулись. Затем мы попросили поработать с прототипом других сотрудников компании, в частности менеджера по продукции и персонал технической поддержки, и также получили их отзывы. Наконец, мы испытали прототип за пределами компании, чтобы узнать, какое впечатление он произведёт на реальных пользователей. Мы выполняли доводку интерфейса очень быстро, прорабатывая до десятка прототипов в неделю. Действительно, во время проведения фазы тестирования нам удалось обнаружить ряд крупных проблем со слиянием данных. Страшно подумать, сколько времени отнял бы поиск и решение этих проблем обычными методами: наверное, не меньше, чем время полного цикла работы над выпуском, а может и больше. За короткое время (в сумме не больше двух недель) мы закончили проектирование интерфейса. Он не был совершенным, в дальнейшем пришлось вносить небольшие изменения. Но ещё до начала написания кода у нас была готовая на 90%, проверенная пользователями конструкция. Проработанный пользовательский интерфейс позволил точнее спланировать реализацию проекта, а тестировщики и создатели документации смогли если не опробовать программу на деле, то хотя бы разобраться в её особенностях раньше, чем она была написана.

Роль специалиста по инженерной психологии

Специалисты по инженерной психологии играют критическую роль в разработке программ. Мне приходилось работать как с ними, так и без них. В результате я бы предпочёл всегда иметь одного или несколько таких специалистов в своей команде. Однако роль специалиста по инженерной психологии не всегда ясна и существенно варьируется в различных компаниях. В следующем разделе описана работа специалистов по инженерной психологии в компании NuMega.

Сфера ответственности

Специалисты по инженерной психологии отвечают за решение следующих задач:

• Формулирование требований к прототипу пользовательского интерфейса

Создание прототипа пользовательского интерфейса — неотъемлемая часть разработки продукта. Специалисты по инженерной психологии руководят составлением требований к прототипам и их дизайном. Хотя это одна из основных обязанностей этих специалистов, отсюда не следует, что они работают в изоляции от команды или пользователей. Напротив, они должны возглавлять процесс создания ПО и воплощать как собственные идеи, так и идеи участников команды.

• Формулирование требований к прототипу программы установки

Программа установки не менее важна для продукта, чем другие его части, и так же нуждается в прототипе пользовательского интерфейса. Специалисты по инженерной психологии играют ключевую роль в обеспечении максимальной простоты установки ПО, упрощая процедуру, насколько возможно, и освобождая её от излишка функций, параметров и ручного труда.

• Формирование первоначального впечатления от продукта

Одна из задач проекта — сделать первоначальное впечатление от продукта положительным. Открыв коробку, пользователь в первые же десять минут без проблем должен установить программу и начать работать (т.е. решать свои проблемы с помощью вашей программы). Если он быстро преуспеет в этом, повышается вероятность того, что он и дальше будет тратить своё время, чтобы полностью освоить продукт и разобраться в его возможностях. Специалисты по инженерной психологии играют ключевую роль в решении этой задачи, беря на себя определение, формирование и оценку первоначального впечатления от продукта.

Из собственного опыта

Специалистам по инженерной психологии NuMega удаётся заметно улучшить первоначальное впечатление от работы с продуктом путём учёта всех его аспектов. Один из предложенных ими способов улучшения первоначального впечатления включает использование быстрой справки, выполненной в виде четырёхцветных карточек с кратким описанием решения ключевых пользовательских задач, в котором применяются снимки экрана, выноски и

короткие тексты. На обороте карточек также размещается краткая справка по основным вопросам работы с программой, которая помогает пользователям в поисках ключевой информации. Справочные материалы печатаются на высококачественной немнущейся и непачкающейся бумаге, что повышает срок их службы. Эти карточки полностью оправдали себя и стали популярным предпродажным материалом.

• Создание графики, изображений, значков и цветовых схем

Специалисты по инженерной психологии должны обладать навыками создания графических материалов для презентации ПО. Хотя подобные умения и не обязательны для этих специалистов, это ещё один довод в пользу найма людей с разносторонними навыками при любом удобном случае.

Консультирование

Практичность ПО часто определяется не только программой, но и упаковкой, лицензионным соглашением и документацией продукта. Хотя эти предметы не входят в сферу их прямой ответственности, специалисты по инженерной психологии консультируют менеджера по продукции, специалистов по сбыту и команды технических писателей по многим ключевым вопросам, к которым относятся следующие:

• Лицензионное соглашение

Прототип лицензионного соглашения также влияет на впечатление, которое производит ПО на пользователей и в конечном счёте на их удовлетворённость работой с продуктом. Группе специалистов по инженерной психологии принадлежит ключевая роль в создании прототипа лицензионного соглашения и обеспечении максимальной простоты и ясности его формулировки, в то же время не подвергая риску задачи менеджеров проекта.

• Упаковка

Упаковка может оставить очень сильное впечатление как о самом продукте, так и о выпускающей его компании. Специалисты по инженерной психологии вместе с менеджерами по продукции вместе создают простую, практичную и эффектную упаковку для вашей продукции. В частности, NuMega важно выглядеть максимально профессионально и не производить впечатление небольшой компании, которой она в действительности является.

• Документация, электронная справка и дополнительные материалы

Специалисты по инженерной психологии также принимают ключевое участие в создании документации, электронной справки и дополнительных материалов. Они обеспечивают соответствие стиля этих материалов стилю продукта и упаковки. В частности, они контролируют подбор графики и цветовой схемы для всех составляющих товара, чтобы внешность продукта была выдержана в едином стиле. Где бы ни использовались логотипы компании, продукта, информация о продукте, стиль должен быть един. Специалисты по инженерной психологии также следят за техническими характеристиками, обеспечивая их согласованность в программе, документации, электронной справке, карточках быстрой справки и маркетинговых материалах.

Исполнение проекта

Ну, хороший прототип пользовательского интерфейса создан, работа над проектом начата и идёт полным ходом. Кажется, миссия специалиста по инженерной психологии закончена? Вовсе нет, работы для него ещё предостаточно. Основные задачи, которые приходится решать специалистам по инженерной психологии во время исполнения проекта таковы:

• контроль хода работы над пользовательским интерфейсом, как текущий, так и по завершении каждого существенного этапа работы над проектом;

• консультации и санкционирование мелких изменений пользовательского интерфейса;

• создание или поиск элементов графического оформления продукта;

• надзор за созданием упаковки, лицензионного соглашения и формирование первоначального впечатления от продукта (см. выше);

• внутренняя и внешняя проверка ПО: хотя для крупных изменений может не остаться времени, небольшие изменения могут быть вполне возможны — это поможет лучше подготовиться к работе над следующим выпуском;

• визуальный контроль за ПО для обеспечения соответствия стандартам платформы, для которой ПО создано.

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Излишняя доводка кода

Одна из наиболее распространённых причин срыва планов разработки ПО заключается в том, что команда пытается воплотить в программном коде каждое улучшение прототипа пользовательского интерфейса. Если излишняя доводка имеет место во время разработки настоящей программы, создание хорошего прототипа заметно замедляется, и дата завершения продукта становится абсолютно непредсказуемой. Второй или третий вариант интерфейса, как правило, принимается независимо от того, хорош он или нет, чтобы наверстать упущенные сроки реализации проекта или из-за того, что вышло отведённое на доводку время. Не устану повторять, что важно как можно скорее довести дизайн интерфейса, не зацикливаясь на совершенствовании его кода, и утвердить окончательный вариант прежде, чем переходить к составлению плана.

Отсутствие отзывов извне

Если спросить у разработчиков, нужны ли им внешние отзывы при разработке пользовательского интерфейса, в ответ почти всегда можно услышать «Да». И всё же в жизни совсем мало команд получает внешние отзывы о своих проектах, особенно в начале работы. Дело в том, что трудно дать отзыв о том, чего ещё нет. Описанная в этой главе методика позволяет более успешно собирать внешние отзывы.

Лишние нововведения

Представленные в этой главе идеи чаще всего критикуют за то, что они, якобы, «душат» инновации. А что, если спустя несколько месяцев работы над продуктом возникла замечательная новая идея? Стоит ли вносить изменения, если есть возможность?

Фундаментальная идея этой главы в том, что основные элементы пользовательского интерфейса должны быть «на местах» уже в начале работы над проектом. Их нельзя существенно изменять, если мы хотим уложиться в первоначальный план. Бесспорно, инновации не только возможны, но и необходимы, однако работу с ними нужно завершить на этапе работы с прототипом. Именно поэтому следует быстро проверять и отрабатывать разные варианты. Протестировав прототип заранее, вы избавляете себя от необходимости вносить значительные изменения в будущем. Даже при возникновении новой идеи, представляющей прорыв в данной области, сохранится уверенность в том, что текущие идеи в состоянии удовлетворить потребности рынка, что позволит уложиться в план. Я не говорю, что во время реализации проекта нельзя вносить небольшие изменения. Как правило, во время разработки возникает масса полезных идей, существенно повышающих ценность программы с очень небольшими затратами. Многие из них вполне могут быть реализованы без риска срыва плана или возникновения серьёзных проблем.

Планирование

Создание плана часто оказывается одним из наиболее затруднительных и насыщенных политикой аспектов проекта. Правильно составленный план становится эффективным средством управления проектом. Как бы усердно ни трудилась команда, при плохом планировании вся работа пойдёт насмарку, если опоздать с выпуском ПО. Сложность планирования проектов общеизвестна, однако именно понимание принципов рационального планирования часто отличает реалистичную оценку сроков реализации проекта от планов «с потолка».

В этой главе мы подробно рассмотрим сведения, необходимые для составления плана, и обсудим важнейшие понятия планирования. Кроме того, я покажу, как создать точный и реалистичный план.

Предпосылки

Прежде чем приступать к планированию, нужно уяснить требования к проекту, особенности технологии, намеченной для использования в нём, и конструкцию пользовательского интерфейса программы (рис. 11-1). Разобравшись в фундаментальных аспектах проекта, можно получить чёткое представление о том, что будет создано и как это будет работать.

Рис. 11-1. Исходные данные, критически важные для процесса планирования.

Однако если основные требования не определены, а самые рискованные фрагменты программы не отработаны на прототипах или моделях интерфейса, то для разработки точного плана просто не хватит данных. Тогда сформулировать все задачи проекта и точно оценить время, необходимое для выполнения каждой из них, будет невозможно. Поэтому весь проект будет составлен из расплывчатых задач, у которых слишком общая формулировка, препятствующая мониторингу и

контролю их выполнения. При этом получится нереалистичный план, от которого придётся отказаться при первых признаках трудностей. В результате вы останетесь без средства управления реализацией проекта.

Основные понятия и трудности планирования

Чтобы создать план проекта или оценить план, созданный другими, нужно хорошо разбираться в основных понятиях планирования. В этом разделе мы обсудим основные понятия, которые должен знать каждый участник процесса планирования. Затем я опишу наиболее серьёзные трудности работы с людьми, возникающих при создании плана. И в завершение мы рассмотрим ряд наиболее распространённых проблем, с которыми сталкиваются команды разработчиков при планировании.

Основные понятия

Следующие понятия являются фундаментальными для создания надёжных планов.

Равновесие

Объём предстоящей работы, количество доступных ресурсов и время, отведённое на реализацию проекта, должны быть сбалансированы — вот старейшее и самое важное правило планирования. Если хоть один из этих параметров начинает перевешивать или, хуже того, наложены ограничения, которые нельзя сбалансировать, создать ПО вовремя будет невозможно. Даже если в начале работы проект был сбалансирован, велика вероятность, что в дальнейшем равновесие будет нарушено. В цикле разработки возникает достаточно препятствий, чтобы вывести из равновесия даже самый лучший план. По ходу работы менеджер проекта должен регулярно проверять план и всемерно поддерживать его равновесие. Способы мониторинга проекта и внесения изменений в процессе его реализации мы обсудим в главе 12.

Задачи и оценка времени для их выполнения

Задачи — это основные строительные блоки плана, они являются представлением конкретной работы, которую нужно сделать. В общем верно, что легче следить за ходом выполнения небольших задач. Кратко и точно сформулированные задачи позволяют быстро обнаружить отставание от плана. Если задачу нельзя завершите за 1-2 недели, её следует разбить на две или больше меньших задач. Исполнение плана, составленного из долгосрочных задач, труднее контролировать.

При составлении списка задач обязательно нужно учитывать их взаимосвязи в рамках проекта. Например, зная, как одни задачи зависят от других, можно расположить их в нужной последовательности, причём ключевые задачи всегда должны завершаться в первую очередь. Нужно выяснить, сколько времени займёт каждая задача. Это можно сделать, оценив время, необходимое для выполнения некоторой задачи (т.е. выдвинув обоснованное предположение о сроках). Поскольку для формулирования требований, конструирования интерфейса и реализации выбранной технологии сделано уже довольно много, должно быть накоплено достаточно информации, чтобы точно и без особых

затруднений оценить срок для выполнения той или иной задачи. Если точно оценить время исполнения ключевых задач невозможно, вы, вероятно, провели недостаточно экспериментов, исследований и работы с прототипом.

Оценка должна учитывать всю работу, необходимую для выполнения задачи. Например, специалисты по ПО должны оценить суммарное время конструирования низкоуровневой структуры, а также реализации, отладки и блочного тестирования программы. Специалисты по обучению пользователей должны оценить, какое время потребуется на написание, рецензирование, редактирование и правку их материалов. Хотя каждый участник команды самостоятельно оценивает срок завершения своей части работы, его оценку всегда должен проверить ведущий специалист в соответствующей области. На основе этих оценок рассчитывается время реализации проекта, поэтому следует быть уверенным в цифрах.

Со временем вы увидите, что ваши оценки становятся все точнее, особенно при работе над аналогичным продуктом с использованием те же самых технологий. Обязательно проанализируйте задачи, на которые ушло значительно больше времени, чем ожидалось. чтобы понять, почему в оценку вкралась ошибка.

Полнота плана

Не совершайте ошибку, планируя лишь разработку самой программы: план должны отражать все аспекты проекта.

• Разработка

Часть плана, регламентирующая разработку ПО, должна отводить достаточно времени на разработку, блочное тестирование и отладку всех функций программы. Команде разработчиков следует выделить дополнительное время на анализ результатов работы тестировщиков и материалов, подготовленных группой по обучению пользователей. Не исключено, что разработчикам также понадобиться время для анализа результатов специалистов по инженерной психологии и технологов.

• Тестирование

Этот раздел должен давать достаточно времени для создания планов и сценариев испытаний, а также для тестирования самой инфраструктуры. План испытаний должен выделять добавочное время на тестирование ПО после окончания каждого промежуточного этапа работы.

• Обучение пользователей

Здесь должно даваться достаточно времени на создание документации, электронной справки и учебника по работе с программой. На редактирование потребуется дополнительное время, которое также нужно учесть. И не забывайте проводить разбор технических особенностей ПК с участниками команды.

• Работа инженерных психологов

Отведённого здесь времени должно быть достаточно для разработки детальной конструкции пользовательского интерфейса и его оценки. Выделите также время на оценку графических материалов продукта, внутренние и внешние испытания пользовательского интерфейса, и проверку документации. Кроме того,

в плане должно быть время для проверки первоначального впечатления от продукта.

• Работа над выпуском

Времени, запланированного в этом разделе, должно хватить для создания системы сборки ПО, разработки установочной процедуры, для конфигурирования и сопровождения системы управления исходным текстом.

• Зависимость от внешних факторов

План должен в полной мере учитывать возможную зависимость проекта от внешних факторов и предусматривать выделение дополнительного времени в случае необходимости. К таким факторам относятся поставки и использование ПО от сторонних разработчиков, доступность оборудования и даже расширение штата или получение поддержки от других групп.

Параллельная разработка

Одна из основных идей этой книги может быть сформулирована так: параллельная реализация всех аспектов проекта повышает эффективность цикла разработки. Для её воплощения прекрасно подходит план проекта. При этом целью является реализация функций ПО путём интеграции различных задач по разработке, тестированию, обучению пользователей, инженерной психологии и работе над выпуском ПО.

Рассмотрим пример. Разработчики должны реализовать функции, соответствующие командам «Создать клиента», «Изменить клиента», «Удалить клиента». Как только эти функции станут готовы и появятся в ежедневной сборке ПО, команда тестировщиков должна испытать их и дать отзыв о качестве реализации этих функций. В то же время группа специалистов по инженерной психологии должна оценить пользовательский интерфейс с точки зрения его соответствия стандартам эргономики, практичности и задачам разработки. Группа по обучению пользователей должна привести описание этих функций к окончательному виду и дать отзыв о качестве их реализации и интеграции.

У параллельной разработки масса преимуществ. Во-первых, она концентрирует усилия всей команды, что позволяет как можно скорее завершить разработку набора функций. Это создаёт у участников команды ощущение срочной целенаправленной работы и (я на это надеюсь) успеха проекта уже на ранних стадиях его реализации. Кроме того, она позволяет сохранять синхронность работы команды в течение всего процесса разработки, так как все её члены обсуждают и решают одни и те же проблемы. Во-вторых, поскольку вся команда сосредоточена на разработке одних и тех же функций, можно будет намного раньше понять, действительно ли завершена функция или пока написан только её исходный текст, страдающий от недостатка качества интеграции и мало пригодный к использованию. Задача в том, чтобы заставить команду как можно скорее создавать надёжно работающие функции, чтобы не возвращаться к проблемам, давно считавшимся решёнными. Не правда ли, было бы очень неприятно получить сюрприз в виде плохого качества или недостатков в реализации функций, считавшихся законченными уже несколько недель, или месяцев тому назад.

Баланс ширины и глубины охвата в работе над проектом

Следует так упорядочивать задачи при создании плана, чтобы группы работали по всему фронту проекта, а не над отдельными его частями. Короче, не ограничивайтесь реализацией какой-либо одной части системы, игнорируя остальные. Например, работая над приложением для размещения заказов через Web, не составляйте план так, чтобы сначала был разработан пользовательский интерфейс, затем реализована прикладная логика, и лишь потом — весь код для работы с базой данных. Даже при наличии детальных спецификаций структуры, поочерёдное решение всех задач обернётся кошмаром при их интеграции, поэтому надо работать над всеми частями системы одновременно. Сосредоточьтесь на решении задачи, которая позволит ввести простой заказ, сохранить его и вывести подтверждение. Такой подход позволит сразу создать комплексное решение, пригодное для тестирования и объединяющее все необходимые программные подсистемы проекта.

Контекст функций

Часто в ответ на вопрос о планах от разработчика можно услышать такое: «Сначала нужно обновить менеджер ресурсов поддержкой 32-разрядных идентификаторов, потом изменить алгоритм анализа индекса PRODUCT ID, чтобы разрешить дублирование записей, а затем переписать обработчик ошибок, чтобы поддерживалась многопоточностъ». Каждый из этих пунктов вполне допустим, как элемент работы программиста, однако следует удостовериться, что все они находятся в контексте функций программы или не выходят за рамки её требований. В контексте некоторой функции задачу разработчика можно сформулировать, например, так: «организовать поддержку печати из диалогового окна ввода заказа» или «обеспечить возможность ввода нескольких заказов одновременно». Более узкая сосредоточенность также полезна для других разработчиков команды, поскольку им важно знать, когда некоторые функции станут доступны, а не срок завершения задач, необходимых для реализации этих функций. Когда план чётко определяет срок завершения всех функций или требований, остальные участники команды могут быть уверены, что их работа завершится в параллели с другими задачами.

Трудности в работе с людьми

Ниже описан ряд трудностей при составлении плана, имеющих отношение к работе с людьми.

Распределение работы

Не все разработчики от рождения наделены равными способностями. Некоторые лучше всего программируют интерфейсы, другие — системную логику. У одних опыта больше, у других — меньше. У некоторых производительность труда очень высока, у других средняя или даже низкая. Нельзя назначать задания случайным образом, полагая, что все люди обладают «типовыми» способностями. Распределяя задания между разработчиками, тестировщиками, технологами и другими членами команды, следует быть очень осторожным. В каждом случае надо учитывать уровень мастерства, индивидуальную производительность, опыт практической работы над проектами и привычки.

Балансировка нагрузки

Задача состоит в равномерном распределении рабочей нагрузки по реализации проекта на основе индивидуальных способностей участников команды. Однако будьте осторожны и не перегрузите лучших участников команды. Хотя они могут сделать больше других, у них тоже есть свой предел. Эти люди ещё пригодятся, чтобы помочь другим, когда возникнут неприятности.

Возможные накладки

Наивно полагать, что все своё рабочее время люди будут трудиться над своими основными задачами. В каждой организации возникают накладки, к которым относится время, потраченное на собрания, наладку технологии, отпуска, стажировки, командировки, больничные и выходные. Даже при 80 рабочих часах в неделю, нетрудно заметить, что 10 из них тратятся на отвлечённые действия. Это тоже следует учесть в плане. Кроме предсказуемых событий (отпусков, командировок и т.п.), план должен учитывать и неожиданные: болезни сотрудников, зимнюю непогоду и пр.

Задачи: критичные и некритичные

Одни задачи критичны для продолжения работы над проектом, другие — нет. Опоздание в выполнении некритичных задач не влияет на ход реализации плана в целом, а задержки с критичными задачами непременно отражаются на реализации плана. Планируя, нужно определить, к какому виду относятся те или иные задачи. Нужно постоянно следить за исполнением критичных задач, так как любая задержка повлечёт за собой срыв конечных сроков плана или рост сверхурочной работы команды. Лучше поручать решение критичных задач опытным людям, чтобы свести к минимуму риски проекта.

Ловушки, подстерегающие любую команду

Ниже описан ряд наиболее распространённых проблем с планированием, с которыми приходится сталкиваться командам разработчиков.

Сроки: конечный и согласованный

Конечный срок —

это предположительная дата сдачи проекта. Обычно он основывается на внешней рыночной конъюнктуре и состоянии дел в отрасли. Эта дата очень важна, поэтому нельзя соглашаться с конечным сроком, не составив прежде план. Подставьте этот срок в «уравнение» планирования, как одну из переменных и попробуйте уравновесить требуемую функциональность ПО и ресурсы для её разработки. Если уравнение не решается, придётся исключить часть функций, добавить ресурсы или сделать то и другое в некоторой пропорции. Конечная цель в том, чтобы составить уравновешенный, реалистичный и правдоподобный план, против которого не стал бы возражать ни один член команды.

Как только появится хороший план, необходимо удостовериться, что в команде нет возражений.

Согласованный

срок — это дата сдачи ПО, с которой согласны все участники команды. Они считают эту дату разумной и вполне достижимой. Таким образом, команда

разработчиков принимает на себя обязательство закончить ПО к этому сроку. Ситуация в небольших начинающих фирмах и крупных компаниях сходна тем, что от своевременного окончания работы над ПО зависит результат работы множества людей и значительных затрат, как денежных, так и временных. Для компании чрезвычайно важно выдержать утверждённый согласованный срок, чтобы, выполнив принятые обязательства, завоевать доверие к своей компании.

Ответственность за реализацию плана

Чаще всего команду ставят перед фактом, жёстко определяя необходимый объём функциональности ПО, выделенные для этого ресурсы и срок, к которому всё должно быть готово. И получается, что ответственность за выполнение плана лежит не на разработчиках, а на организации или персоне, которая эти требования «спустила сверху». Такова общая формулировка этой серьёзной проблемы. Боевой дух участников команды будет невысок: ведь они будут чувствовать, что их поставили в заведомо проигрышное положение. Без чувства ответственности, не принимая на себя обязательств, команда не сможет вложить в реализацию проекта сердце и душу, и никакого энтузиазма.

Вместо этого группа разработки должна создать свой собственный план, точнее, сама поддерживать баланс в рамках плана. С принятием обязательств в команде появляется чувство ответственности. Выдвинув свой план разработки ПО, за который она отвечает, команда должна приложить все усилия, чтобы выдержать установленные в нём сроки. Доверие — это следствие выполненных обязательств.

Из собственного опыта

Разработка ПО в NuMega обычно проходила под огромным давлением необходимости уложиться в срок. Конечные сроки сдачи наших продуктов обычно приурочены к выходу Microsoft Visual Studio или появлению новых платформ и технологий, например Microsoft Windows 95, Microsoft Windows NT или Microsoft COM. Чтобы воспользоваться преимуществом этих событий, наши группы маркетинга разработали всесторонние планы продвижения продукта, включающие рекламу, пресс-конференции, аналитические исследования, презентации и обучение продавцов. Ассигнования на эти мероприятия, зависящие от даты выхода ПО, достигают сотен тысяч долларов. Кроме того, наши специалисты по продажам и старшие менеджеры рассчитывали на существенный прирост прибылей с выходом каждой последующей программы. Любая задержка была чревата не только потерей больших денег и времени, но и упущенными возможностями по продаже и потерей выгодной для нашего товара рыночной конъюнктуры.

Чтобы обеспечить своевременный выпуск ПО, вся «домашняя работа» (поиск компромиссов между реализацией функций, доступным временем и ресурсами) выполнялась заранее, затем на основе конечного срока выхода ПО составлялись реальные планы. Таким образом, автором планов были технические специалисты, а не экономисты или старшие менеджеры. Приходилось брать на

себя ответственность за реализацию этих планов независимо от их содержания. Любая ошибка планирования была нашей проблемой, и мы отвечали за то, чтобы найти решение, не допуская задержки выпуска ПО.

Вопрос доверия к техническим специалистам

Одна из наибольших проблем, с которыми сталкиваются технические специалисты, — нехватка доверия. Постоянно нарушая сроки, техническая группа теряет доверие остальных подразделений организации. Это угрожает потерей доверия к плану и достоверности суждений о возможных компромиссах проекта, а также открывает лазейку в планировании для разного рода игр («липовым» срокам сдачи, заведомо завышенным просьбам в расчёте получить хотя бы часть от запрошенного). Однако хорошая репутация, завоёванная своевременным исполнением работы, — источник доверия, которое при необходимости позволяет бороться с серьёзными трудностями, привлекать дополнительную поддержку и извлекать выгоду из чужих сомнений.

Как составить хороший план

Теперь можно сосредоточиться на особенностях составления хорошего плана разработки ПО. В этом процессе три основных этапа: определение задач, объединение задач в группы, называемые базовыми уровнями, и группировка последних в этапы проекта.

Ниже я приведу пример типичного плана с описанием основных структур проекта, необходимых для разработки плана. Легко заметить, что целью планирования не является «микроуправление» каждой деталью проекта. Просчитать, чем будет заниматься каждый член команды в течение шести месяцев, начиная с сегодняшнего дня, скорее всего невозможно. Вместо этого нужно составить план со списком чётко определённых задач, плавно сменяющих друг друга по ходу цикла разработки. Такой план позволяет контролировать работу команды над каждой задачей, а также отслеживать ход реализации проекта со значительной определённостью.

Задачи

На первом этапе следует определить все задачи, решение которых позволяет реализовать некоторую функцию. Суммарное время выполнения этих задач составляет общее время реализации функции. Сначала надо спланировать реализацию необходимых функций и лишь затем переходить к планированию желательных и возможных функций. Такой метод позволит как можно скорее получить жизнеспособную программу.

Затем следует распланировать задачи групп тестирования, обучения пользователей, разработчиков пользовательского интерфейса и технологов. У каждой группы должен быть свой набор задач, определяемый на основе частей проекта, за разработку которых отвечает группа. Эти задачи должны быть организованы так, чтобы их можно было интегрировать и их реализация не слишком отставала от реализации функций разработчиками.

Базовые уровни

Базовые уровни определяют срок реализации группы связанных функций. Каждые 2-3 недели должен быть готов очередной базовый уровень. Помните: соответствующий фрагмент ПО должен устанавливаться с помощью программы установки, а его функциональность должна быть доступна для разработчиков. Реализация базовых уровней — важные краткосрочные цели, на достижении которых необходимо сосредоточить внимание и усилия команды. Вообще ничто не может быть важнее своевременного завершения очередного базового уровня. Если он запаздывает, можно официально говорить об отставании проекта от плана, что требует немедленных корректирующих действий. Ниже приводится ряд базовых уровней (из продукта BoundsChecker, разработанного NuMegaдля обнаружения ошибок в программах).

• Создана библиотека исходных текстов, выполнена первая ежедневная сборка программы, закончена установочная процедура для «скелета» программы.

• Программа позволяет активизировать основные функции для работы с памятью и вести регистрацию их работы.

• Программа выводит первое сообщение об ошибке при работе с памятью с помощью прототипа пользовательского интерфейса.

• Программа успешно обнаруживает утечки памяти типа 1 и 2.

• Программа успешно обнаруживает утечки памяти типа 3 и 4.

• Программа успешно обнаруживает утечки памяти типа 5 и 6.

• Появляется реальный пользовательский интерфейс программы, но без поддержки печати, сортировки и фильтрации.

• Закончена поддержка печати, сортировки и фильтрации.

• Программа интегрируется с другими программами пакета.

Промежуточные этапы

Промежуточный этап — это группа базовых уровней, представляющих законченную часть программы. Необходимо равномерно распределить их завершение по ходу работы над проектом. Например, если для проекта определено 4 промежуточных этапа, то каждому из них должны соответствовать 25% реализации проекта. Очевидно, что чем сложнее проект, тем больше у него промежуточных этапов.

У каждого промежуточного этапа должен быть период стабилизации и интеграции (см. главу 6). Напоминаю, что в это время (обычно 1-2 недели) вся команда концентрируется на решении проблем, обнаруженных в реализованных функциях. Периоды стабилизации жизненно важны для проекта, поскольку в это время проводится тестирование, исправление ошибок, устранение неполадок в структуре и интеграции, проводится оценка производительности, т.е. все мероприятия, способствующие стабилизации программы. Не приступайте к реализации новой функции, пока не убедитесь, что только что законченные функции работают хорошо. Помимо всего прочего, периоды стабилизации очень удобны для разного рода доработок. В это время отставшие участники или подразделения команды могут наверстать упущенное и догнать остальных, чтобы вновь работать синхронно.

Внешние промежуточные этапы

В завершении внешних промежуточных этапов участвуют люди или группы, не работающие над проектом постоянно. Внешние промежуточные этапы знаменуют собой критические точки проекта. Вот самые распространённые внешние промежуточные этапы:

• альфа-версия — выпуск, в котором реализован лишь ряд критических функций программы; альфа-версии не предназначены для широкого использования, однако могут быть полезны для демонстрации прогресса проекта или сбора внешних отзывов о работе критических функций;

• бета-версия — выпуск, в котором реализованы если не все, то большинство функций; бета-версии передаются клиентам для испытаний и оценки;

• кандидат на выпуск — в случае успешного окончания тестирования этот выпуск будет передан в производство для тиражирования; появление кандидата на выпуск — знак того, что проект почти закончен и выпуск ПО состоится;

• передача в производство — к этому времени рабочий выпуск будет передан в производство для тиражирования (или опубликован в Web, в зависимости от назначения ПО).

Любой из внешних промежуточных этапов требует распространения ПО за пределами команды или даже компании. Так как это очень важное событие, перед каждым промежуточным этапом нужен период стабилизации. Он позволяет команде сосредоточиться на его качестве, интеграции; выполнить «подгонку» частей и устранить оставшиеся неполадки перед выпуском продукта. (Подробнее о бета-версии и кандидате на выпуск см. главы 13 и 14)

Пример

Чтобы закрепить основы, давайте шаг за шагом рассмотрим подробный пример (табл. 11-1). В таблице показано упрощённое описание проекта, откуда удалена часть информации, обычно имеющейся в нём. Тем не менее, этот пример достаточно детализирован, чтобы продемонстрировать стыковку всех частей проекта. Ниже приводится ряд допущений, сделанных при планировании этого примера.

• Принципы планирования.

* С каждой функцией связан список технических задач. В этом примере они не показаны, однако их легко перечислить в реальном плане. На выполнение одной задачи отводится не более 2 недель, а на большинство — неделя или даже меньше. В зависимости от приоритетных требований к ПО, в первую очередь реализуются необходимые функции, а затем — менее важные.

* Тестирование функций осуществляется по мере завершения их разработчиками. Тестирование некоторых функций будет автоматизировано, другие же придётся тестировать вручную. Подробное описание испытаний приводится в плане тестирования.

* Специалисты по обучению пользователей составляют описания функций по мере их завершения. Работа по составлению документации должна как можно меньше отставать от реализации функции. Подробно эти действия описаны в плане обучения пользователей.

* Специалисты по инженерной психологии оценивают качество реализации всех функций пользовательского интерфейса, консультируют по поводу внесения изменений и контролируют впечатление от продукта по мере реализации проекта. Детали работы специалистов по инженерной психологии описаны в специальном плане.

* Во время работы над выпуском ПО сразу же создаётся простая сборка программы и установочная процедура. Затем сборка и установочная процедура будут регулярно пополняться новыми функциями, таким образом, они будут включать все большую долю функциональности готовой программы. Они также поддерживают подключение новых функций по мере их готовности. Конкретные усовершенствования функций и возможностей программы будут описаны в плане работы над выпуском.

• Участники работы над проектом:

— Мэтт — ведущий разработчик, занятый полное рабочее время;

— Джон — программист;

— Джим — ведущий тестировщик, также отвечает за автоматизацию;

— Фрэнк — тестировщик, исполняет автоматизированное и ручное тестирование функций;

— Сара — ведущий специалист по обучению пользователей;

— Кенни — ведущий специалист по инженерной психологии;

— Боб — ведущий технолог.

• Промежуточные этапы, внешние и внутренние.

— План проекта состоит из 4 базовых уровней, на реализацию которых отводится по 2 месяца, и 2-х главных промежуточных этапов. Будут выпущены 2 бета-версии, 1 версия — кандидат на выпуск и 1 версия для тиражирования.

— Каждый промежуточный этап образован 2 базовыми уровнями. Первому промежуточному этапу будет соответствовать наполовину законченный проект, а второму этапу — полностью законченный проект.

— Работа над бета-версией 1 займёт 1 месяц. Функции 14 и 15 будут добавлены во время работы над бета-версией 1, а оставшееся время будет потрачено на тестирование, настройку и исправление ошибок. У каждого участника группы есть некоторый список действий на время работы над бета-версией 1.

— В бета-версии 2 не будет новых функций по сравнению с бета-версией 1. Внесение значительных изменений в главные функции не допускается, разрешено лишь тестирование, настройка и исправление ошибок. У каждого члена группы есть список задач на это время.

• Кандидат на выпуск.

Версия — кандидат на выпуск будет готова к концу работы над бета-версией №2, если её тестирование пройдёт успешно и не будет обнаружено серьёзных ошибок.

• Контрольные собрания.

— Проведение собраний для контроля за состоянием проекта запланировано на каждый понедельник. Если достигнуть базового уровня вовремя

не удалось (или все говорит об этом), придётся вносить изменения, чтобы наверстать упущенное.

Табл. 11. Примерный план.

Цифрами обозначены функции, состояние функций обозначено следующими буквами:

Ф — функция запрограммирована, выполнено блочное тестирование и завершены все связанные с ней технические задачи;

А — тестирование функции автоматизировано:

Р — функция протестирована вручную;

Д — функция документирована:

И — проверена простота использования функции.

Добавления в бета-версии

Легко заметить, что реализация двух задач в этом примере запланирована на период работы над бета-версией 1. Во время работы над любой бета-версией надо воздерживаться от добавления новых функций, особенно важных и сложных. Однако иногда есть смысл планировать включение в первую бета-версию функций, реализация которых не требует больших затрат и не вносит особого риска. Чем дольше задерживается передача программы в руки бета-тестеров, тем больше времени займёт сбор отзывов о качестве реализации и работе функций программы. Часто польза от раннего цикла бета-тестирования превышает риск включения небольших функций после начала программы бета-тестирования.

Хотя включить несколько дополнительных функций время работы над первой бета-версией ещё допустимо, в период работы над последней бета-версией реализацию дополнительных функции лучше не планировать. При работе над последней бета-версией функции остаются неизменными, и усилия команды концентрируется на качестве, производительности и интеграции. (О тестировании бета-версий см. главу 13.)

Неожиданные проблемы

Создавая план согласно описанным в этой главе принципам, вы, вероятно, будете планировать проект, как обычно. Однако разработка ПО — это не точная наука, и до проблем всегда рукой подать. Чтобы заметить малейшее отклонение проекта от намеченного пути, нужно регулярно проверять ход выполнения плана и после завершения каждого промежуточного этапа сравнивать фактическое состояние проекта с планом. Если работа отстаёт от плана, надо определить проблему, изменить план и постараться завершить очередной промежуточный этап в срок. Все так просто? Однако это та самая ситуация, когда от менеджера проекта и ведущих специалистов требуется полная самоотдача. Поскольку очень сложно вести работу над проектом, не отставая от графика, в третьей части основное внимание уделено именно этому вопросу.

Типичные проблемы и их решение

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик а также их решения.

Ничего не получается!

Создание хорошего плана требует серьёзных усилий, поэтому легко понять, почему некоторые группы используют планы без особого энтузиазма. Бывало, что, пересилив себя, некоторые люди всё же пытались спланировать проект, но часто из-за множества неожиданных проблем им приходилось оставлять план. Если план не отражает действительность, он будет в значительной степени игнорироваться.

Бесспорно, планирование — задача не из лёгких, однако решить её необходимо, поскольку план — это руководство к реализации проекта. Если вы намерены завершить программу в срок, нужно уяснить объём предстоящей работы и время, нужное для её выполнения. Описанные в этой книге методы упрощают планирование, делают его более предсказуемым и менее рискованным.

Это сложнее, чем кажется на первый взгляд…

Кажется, что втиснуть задачи в рамки 1-2 недель так просто, но это может быть настоящим испытанием. Наверное, в плане всегда найдётся задача, на выполнение которой отводится 3-5 недель. Такую долгосрочную задачу лучше разбить на несколько меньших. Чем чаще контролируется прогресс проекта, тем проще обнаружить отклонения от плана. Как правило, трудности при разбиении задачи возникают из-за неполного её понимания, а это тревожный признак. Чтобы лучше разобраться в том, что предстоит сделать, решение нужно смоделировать.

Потеря согласованности

В плане должны быть периоды синхронизации. Помимо стабилизации ПО, они позволяют остальным участникам команды наверстать упущенное/ Нельзя предусмотреть все проблемы заранее, но некоторые из них можно предсказать. Какими бы они ни были, в плане надо выделить время для их решения.

Завершение проекта

Итак, основное планирование закончено — остаётся лишь «нажать на рычаг и выдать готовый продукт». Хотя процесс выглядит довольно механистичным, всё равно приходится постоянно следить за ходом реализации проекта и бороться с повседневными проблемами. В этой главе мы обсудим, как эффективно следить за состоянием проекта и какие меры принимать, чтобы не дать проекту отклониться от намеченного пути.

Аналогия с самолётом

Представим самолёт, следующий из Бостона в Сан-Франциско. Во время полёта бессчётное множество факторов могут нарушить график рейса или сбить самолёт с курса. Однако большинство самолётов, летящих прямыми рейсами, всё же прибывает в пункт назначения по расписанию и приземляется там, где нужно. Как и самолёт, проект нуждается в навигационной системе и управлении в полёте, чтобы не сбиться с курса и не нарушить график. Без такого механизма работа над проектом будет похожа на полет вслепую.

•

План полёта

План полёта разрабатывается задолго до взлёта. Среди прочего в плане описаны этапы полёта, следуя которым экипаж может привести самолёт из пункта отправления в пункт назначения (например: «лететь 100 миль на запад, в пункте X повернуть на северо-запад, затем 500 миль держать этот курс» и т.д.).

Аналогично работа с прототипом и моделью использования ПО позволила сформировать базовое представление о том, что создаётся в рамках проекта и на что это будет похоже в готовом виде. Грамотно проведённое планирование позволило наметить основные этапы создания продукта и определить сроки их выполнения. Таким образом, основное внимание во второй части этой книги (главы 12, 13,14) мы сосредоточили на создании «плана полёта» для проекта.

•

Элемент непредсказуемости

Хотя план полёта и регламентирует действия экипажа во время рейса, он не может и не должен предсказывать или решать все проблемы, которые могут возникнуть в полёте. Направление ветра, турбулентности атмосферы, колебания тяги двигателей, пробки в воздушных эшелонах и масса других факторов влияют на полет и потенциально могут отклонить самолёт от курса и выбить его из расписания.

Проект тоже столкнётся с тысячами проблем, которые нельзя ни предсказать, ни запланировать. Любая из них может сбить его с курса или стать

причиной задержки. Как и самолёту, проекту нужна система быстрого реагирования на события, происходящие во время полёта.

•

Навигационная система

Удерживать курс самолёта помогает навигационная система, которая постоянно (примерно каждые несколько минут) определяет его координаты и сравнивает результат с координатами пункта назначения. В полёте навигационная система постоянно вносит небольшие коррективы, чтобы вернуть самолёт на намеченный курс. Принцип работы навигационной системы состоит в допущении, что самолёт всегда летит в неверном направлении, и его нужно вернуть на правильный курс (рис. 12-1).

Рис. 12-1. Навигация в непредсказуемых условиях.

Проекту тоже нужна «навигационная система». Нужно почаще проверять состояние проекта и вносить небольшие коррективы, прежде чем он успеет сильно отклониться от намеченного курса. В ходе разработки надо регулярно оценивать прогресс проекта и своевременно вносить соответствующие изменения.

•

Снижение

По мере приближения к пункту назначения начинается работа по подготовке самолёта к посадке, которая включает ряд специальных действий — от выделения взлётно-посадочной полосы диспетчером аэропорта до поднятия столиков пассажирами. Члены экипажа знают, какие процедуры нужно выполнить, чтобы безопасно посадить самолёт в нужном месте и точно по расписанию. Им не приходиться лихорадочно соображать, что делать, когда самолёт уже пошёл на снижение.

Проект также нужно безопасно «посадить», не допуская никаких промахов при приближении конца пути. Надо организовать процесс плавного «снижения», направляющий завершающие этапы проекта. Этот процесс нужно спланировать заранее, а не создавать в спешке во время «захода на посадку». Из-за важности последнего этапа я решил полностью посвятить ему главу 14.

Процесс измерений и мониторинга состояния проекта

Как сказано в главе 9, план проекта — это совокупность описаний отдельных его этапов, каждый из которых харак-теризуется определённым уровнем законченности некоторых функций программы, который должен быть достигнут к заданному сроку. Эти этапы можно формально рассматривать как

контрольные точки для оценки прогресса проекта. Если функциональность программы реализуется в срок, в заданном объёме и работает должным образом, значит, вы не сбились с курса. Обратная ситуация является проблемой, которую нужно сразу решать. Если программа собирается каждый день, в любой момент её можно установить и её тестирование ведётся параллельно разработке, значит, у вас есть необходимые инструменты и процедуры для регулярного контроля проекта. Кроме того, обладая планом, в котором отмечены даты и параметры контрольных точек, всегда можно определить, на каком этапе находится проект в данный момент. Сравнив текущее положение проекта с планом, можно узнать, в верном ли направлении идёт работа.

Внесём ясность в значение понятий, рассмотренных выше. Если работа над проектом только начата и через две недели должен быть закончен первый этап, во что бы то ни стало нужно закончить его вовремя. Если вы всерьёз собираетесь уложиться в конечные сроки, следует устанавливать промежуточные сроки и выдерживать их. Если для этого придётся работать по ночам и по выходным — работайте. Не дожидайтесь конца работы над проектом, чтобы наверстать упущенное: тогда будет уже слишком поздно. Следует успевать выполнить намеченное к определённому сроку прямо перед наступлением этого срока. Удалось уложиться в срок — это хорошая работа, и успех обязательно нужно отметить всем вместе. Но если работа просрочена, соберите группу, выясните, в чём дело, и устраните проблему. Также необходимо составить план навёрстывания упущенного. Чтобы закончить проект к намеченному сроку, группа должна работать очень серьёзно и напористо, выдерживая промежуточные контрольные сроки.

А есть ли способ заранее узнать, удастся ли достичь следующую контрольную точку вовремя? Завершение отдельных этапов проекта — формальные события, с которыми обычно связаны объективные параметры. Но их, как правило, разделяет несколько недель, а для мониторинга проекта в периоды между соседними контрольными точками нужны дополнительные инструменты.

Решению этой проблемы и будет посвящена остальная часть главы. Я расскажу о правилах сбора текущих сведений о проекте и о том, как при необходимости менять направление и темп проекта. Помните: срыв плана в конце работы над проектом случается не вдруг, а назревает потихоньку, день за днём.

Определение состояния проекта

Сбор и распространение информации между участниками группы — ключ к эффективному исполнению плана проекта. В этом разделе я покажу, как лучше всего собрать сведения о состоянии проекта и довести их до сведения каждого, не таская всю группу по собраниям из-за мелочей, и избежать различных накладок. И, что важнее всего, вы узнаете, как с помощью собранной информации увидеть проблемы до того, как они станут причиной крупных задержек или существенных трудностей.

Ежедневные сборки и базисные тесты

Как сказано выше, ежедневные сборки программы и базисные тесты — это пульс проекта. Оба мероприятия критичны для определения состояния проекта.

Если в течение нескольких дней или недель вы не в состоянии скомпоновать программу, проект в беде. Возможность сборки ПО нужна для поддержания его внутренней согласованности, интеграции и визуализации изменений. Если нельзя выполнить сборку программы, оценить состояние проекта также невозможно.

Кроме того, необходимо проводить базисные тесты, критичные для регулярной (ежедневной) оценки базового, качества ПО. Обнаруженные проблемы надо сразу решать, поступать иначе — то же самое, что сидеть сложа руки, когда самолёт быстро снижается.

Собрания

В той или иной форме контрольные собрания проводятся почти в каждой группе. Контрольные собрания — замечательный способ сбора и распространения ключевой информации о состоянии проекта, поддержания контактов и совместной работы в коллективе. Но если контрольные собрания проводятся плохо, они могут до смерти наскучить людям, породить ощущение беспомощности и нарушить сплочённость группы. Ниже перечислен ряд правил, придерживаясь которых, можно извлечь из контрольных собраний максимум пользы.

- У собрания должна быть определённая цель

На контрольном собрании участники группы сообща обсуждают крупные достижения, неудачи и трудности. Сосредоточьтесь на том, что удалось и что не удалось. На собрании также поднимают важные проблемы, требующие внимания одного или нескольких участников группы.

- На собраниях должны быть все

На контрольном собрании должны быть все участники реализации проекта. К ним относится основной состав группы (см. главу 4): менеджер проекта, разработчики, тестировщики, специалисты по обучению пользователей и технологи.

- Не посвящайте собрание решению частных проблем

Собрания часто становятся местом встречи нескольких участников группы, которые хотят обсудить свои проблемы. Несомненно, эти проблемы важны и требуют разрешения, но контрольное собрание — не место для «мозговой атаки», решения технических и других серьёзных вопросов. Если попытаться охватить проблемы всех участников группы, то лучшую часть дня придётся потратить на собрания, при этом значительная часть времени остальных участников группы будет потеряна зря: люди должны знать суть решения, а не его подробности.

Должен ли каждый разработчик отсиживать на собрании, посвящённой новому формату электронной справки? Должен ли каждый тестировщик выслушивать, как разработчики обсуждают набор изменений в API? Нет. Если проблему нельзя решить за две минуты, для её решения надо провести отдельное собрание, пригласив только тех, кого эта проблема касается. Решения, выработанные на собраниях, посвящённых частным проблемам, следует доложить на общем контрольном собрании. Не используйте общегрупповые контрольные собрания для решения частных проблем.

Столь же неправильно докладывать на контрольном собрании о всех задачах, над которыми пришлось работать за неделю. Нужно лишь сказать, идёт ли разработка проекта по намеченному пути. Если нет, надо перечислить причины, всё остальное несущественно.

- Не затягивайте собрание

Контрольные собрания должны быть краткими и проводиться чаще. Рекомендуется проводить их хотя бы раз в неделю, предоставляя каждому участнику группы пять минут на доклад о состоянии дел в вверенной ему области с учётом вышеописанных требований. Организатор собрания (обычно это менеджер проекта) должен поддерживать обсуждение в определённом русле и не позволять ему переходить на отвлечённые темы. Однако все важные вопросы, поднятые любым участником группы, следует внести в список проблем проекта.

- Ведите список нерешённых проблем

Во время реализации проекта обычно возникают проблемы, которые надо решать. Чтобы не забыть о них, в главе 5 я рекомендовал регистрировать такие проблемы, назначать ответственных за их решение и обязательно разъяснять найденное решение другим. Аналогичную концепцию можно применить при проведении контрольных собраний. На собраниях проводится анализ нерешённых проблем, назначаются ответственные за их решение и устанавливаются сроки. Таким образом, каждый будет знать, что все проблемы будут рассмотрены и решены.

Из собственного опыта

В преддверии завершения важного промежугочного этапа сотрудники компании NuMega должны были сообща устранять последние неполадки и ошибки. Чтобы гарантированно обеспечить себя актуальной информацией, мы

составили список неполадок, упорядоченный сначала по их приоритету, а затем — по именам ответственных за их устранение. Каждый участник контрольного собрания получал копию этого списка (позже печатные списки были заменены подключёнными к сети лэптопами, что позволило работать прямо из системы). Такие обзоры позволили мобилизовать всю группу на решение важных проблем и устранение ошибок, сдерживавших работу других или бывших источником особого риска. Интенсивное общение имеет очень большое значение на завершающих стадиях работы над проектом, бета-версией или новым выпуском программы.

«Управление мимоходом»

Очень полезна технология «управления мимоходом» (Managing by Walking Around, MBWA). Контрольные собрания — формальность, но менеджер проекта и ведущие специалисты, даже просто находясь в группе, могут встречаться с участниками для неформального обсуждения тех или иных проблем.

• Каждый видит, что менеджер проекта участвует в работе и заботится об успехе проекта. Он не собирается отгородиться от исполнителей проекта цифрами, графиками и рисунками, управляя проектом лишь через компьютер. При таком подходе менеджер проекта и ведущие специалисты становятся доступнее, и с ними можно быстро и без проблем обмениваться информацией.

• Люди часто ощущают дискомфорт, выступая на контрольных собраниях. Удивительно, что во время беседы тет-а-тет за чашкой кофе они высказывают такие мысли, о которых скорее всего даже не заикнулись бы на групповом собрании.

• Такая практика допускает спонтанное обсуждение важных предметов и проблем, в ходе которого часто рождаются совершенно новые подходы к их решению. Замечательные идеи не рождаются в два часа пополудни каждый понедельник. Чтобы они возникали, нужно поощрять внеплановое, неформальное и нерегламентированное общение.

• Поскольку почти все ведущие специалисты когда-то были просто инженерами, они очень не любят отрываться от компьютера и покидать свой офис. Но происходят просто удивительные вещи, когда кто-то мимоходом спрашивает их: «как идут базисные тесты?», «что нового с проблемой X?» или «ну как, уже заканчиваем второй этап?»

• Хороший менеджер проекта и ведущий специалист обязательно проводит некоторое время наедине с участниками группы. Такие встречи не обязательно должны быть формальными или проводиться по расписанию. Одного короткого визита участника по какому-нибудь личному или рабочему вопросу достаточно для поддержания сосредоточенной и слаженной работы. Кроме того, это замечательный способ избавиться от проблем с кадрами или иных трудностей при реализации проекта прежде, чем они возникнут. Если вы не хотите получить неприятный сюрприз во время контрольного собрания или, что ещё хуже, накануне сдачи проекта, как можно чаще общайтесь со всеми участниками группы и с каждым по отдельности.

Обмен информацией

Менеджер проекта и ведущие специалисты должны эффективно взаимодействовать друг с другом и с другими членами группы, поскольку совместное переживание главных успехов и неудач имеет решающее значение для прогресса проекта. Поскольку как успехи, так и неудачи одинаково важны, рассмотрим те и другие.

•

Успехи

Отмечайте каждый крупный успех и делайте его достоянием группы. Это важное доказательство того, что проект живёт и здравствует. Поэтому, создав ежедневную сборку программы, первый вариант установочной процедуры или закончив важную функцию ПО, не забудьте разослать участникам группы поздравления по электронной почте, разделив со всеми это радостное событие. В торжества по поводу особо важных событий необходимо вовлекать большую часть коллектива: отделение или даже весь персонал компании.

•

Неудачи

Неудачи необходимо как можно скорее обнаруживать, сообщать о них всем участникам команды и устранять. Не стоит ожидать от команды абсолютно безупречной работы. Знайте: проблемы возникнут непременно. Работа команды заключается в том, чтобы как можно скорее найти их и решить. Худший способ борьбы с проблемами — замалчивать их или вовсе не признавать их существования, так как это не поможет ни решить проблему, ни вернуть проект на намеченный путь.

Чтобы не допустить этого, менеджер проекта и ведущие специалисты должны быть доступны и открыты для общения. Если у члена команды возникает ощущение, что неудачи или проблемы не получат профессионального решения, сокрытие проблемы и неверие в её существование станет хроническим источником бед.

Внесение изменений

Беспроблемных проектов не бывает. Конечно, есть надежда, что «навигационная система» заранее предупредит о трудностях. Это поможет ликвидировать их, прежде чем они перерастут в серьёзные проблемы. Однако чтобы устранить отклонение проекта от намеченного пути, потребуется изменить направление работы и, возможно, увеличивать её темп. Посмотрим, как можно это сделать…

Смена курса

Анализируя возможность изменения существенных элементов проекта (функций, технологии, платформ или плана реализации), нужно обязательно придерживаться следующих правил. Они помогут избежать неприятностей и принять правильное решение.

- Собирайте факты, но не перестарайтесь с их анализом

Часто решение принимают на основе впечатлений, эмоций или единичного случая, а не анализа набора неопровержимых фактов. Прежде чем вносить в проект изменения, убедитесь в их абсолютной необходимости. В частности, не позволяйте эмоциональным утверждениями типа «Программа виснет на каждом шагу» стать причиной отказа от реализации половины функций программы и переброски дополнительных ресурсов на тестирование. Прежде чем действовать, соберите факты. При каких условиях происходит зависание? Кто сообщает о зависаниях? Как часто они происходят? Вполне возможно, выяснится, что все эти зависания связаны с заурядными ошибками, устранёнными ещё на прошлой неделе.

С другой стороны, постарайтесь не попасть в ловушку «паралича анализа». Не убивайте недели на анализ проблем лишь затем, чтобы оказалось, что момент для исправления упущен. Например, если после долгих раздумий так и не удалось собрать достаточно данных, чтобы решиться на изменение какой-то функции, примите решение немедленно, исходя из текущих знаний.

- Вовлекайте в обсуждение проблемы других специалистов

Собрав факты, обязательно обсудите проблему с ключевыми специалистами группы, включая разработчиков, тестировщиков, специалистов по инженерной психологии, технологов и менеджеров по продукции. Проводите «мозговые штурмы», проверяя различные идеи, и обсуждайте альтернативы. Если решение касается других членов команды, дайте им шанс поучаствовать в обсуждении. Приняв решение (даже если решено ничего не предпринимать), известите о нём каждого. Держите команду в курсе всех важных изменений и их причин, а также планов действий на будущее. Плохая информированность об изменениях проекта ведёт к падению боевого духа коллектива.

- Используйте помощь других групп при разработке и тестировании

Часто, когда возникает необходимость что-то добавить, подправить или проверить, все участники группы, как назло, оказываются по уши занятыми своими делами и ни у кого просто нет времени. В этом случае нужно подумать о привлечении дополнительных сотрудников. Если есть свободные люди из отдела технической поддержки, справочного бюро, специалисты по выпуску или

продаже или другие члены команды по созданию продукта, обязательно попросите их помочь.

Из собственного опыта

В NuMega группа технической поддержки часто используется для подстраховки при разработке программ. В службе поддержки компании много ярких личностей, которые хотели бы повысить свой опыт разработчиков. Мы дали им возможность поработать над созданием самых настоящих программ, а они в свою очередь помогли группе разработчиков не выбиться из плана. Естественно, такая помощь часто означала для членов группы технической поддержки сверхурочную работу, но они почти всегда были готовы потратить несколько лишних часов, чтобы помочь разработчикам и приобрести дополнительный опыт. Кроме того, это возможность сделать карьеру. Благодаря полученному опыту и оказанной ими помощи, участники группы технической поддержки завоевали уважение разработчиков и энергично продвигались на должности разработчиков. Это ещё одна причина нанимать стоящих людей на любую должность.

- Нанимайте консультантов и исполнителей

Следует подумать о найме консультантов и исполнителей для выполнения чётко определённой и относительно краткосрочной работы. Ваша задача — закрыть критические участки проекта, на которые не хватает собственных кадров. Но остерегайтесь использования консультантов в качестве ведущих специалистов, так как они, скорее всего, покинут группу по завершении проекта.

- Лучше отказаться от части функций, чем расширятъ план

При необходимости подумайте об отказе от некоторых функций программы. Если приоритетные функции определены и их реализация запланирована на ранних этапах работы над проектом, вполне можно отказаться от ряда второстепенных функций. Прежде чем решиться на расширение плана, рассмотрите возможность исключения ряда функций второго или третьего плана. Отказ от реализации некоторых функций позволяет разгрузить все группы и снизить риск срыва. Расширение плана лишь повышает риск срыва даты выхода продукта. Тривиальное добавление новых функций, не имеющих значения для

успеха ПО — не самое мудрое решение: важнее вовремя выпустить продукт, чем реализовать все второстепенные функции.

Задавайте верные вопросы

Есть хороший способ решить, вносить или не вносить изменение. Для этого нужно спросить у себя: «А что, если я не стану этого делать?» Такие вопросы особенно полезны в борьбе с так называемым «дрейфом функций», который может существенно прибавить работы группе. Однако зачастую изменения вносят без солидного технического или экономического обоснования. Изменение может воплощать хорошую идею, но она может не стоить того риска, которому подвергает весь проект. Конечно, должен быть определённый уровень совершенствования функций, но не до такой степени, чтобы проект был захлестнут потоком мелких изменений. Всегда спрашивайте себя: «А что, если я не стану этого делать?» Это заставит сравнить цену отказа от изменения с ценой его реализации.

Вот примеры хороших вопросов, на которые нужно ответить при поступлении предложения о реализации новой функции.

• Какая часть прибыли будет потеряна, если отказаться от реализации новой функции?

• Скольким клиентами будет полезна новая функция?

• Не сорвёт ли (или подвергнет риску) новая функция срок выхода продукта?

• Снизится ли конкурентоспособность продукта без этой функции?

• Какому риску подвергнется качество продукта при отсутствии новой функции?

• Какое влияние окажет новая функция на использование программы, документацию, а также на процессы её сборки и установки?

Это не означает, что нужно отказаться от изменений вообще, — просто всегда нужно быть уверенным, что выгода от изменения намного больше расходов на её реализацию.

Стремление к согласию не должно мешать принятию решений

Решать проблемы нужно как можно скорее, не позволяя им долго оставаться открытыми. Так или иначе, решение должно быть принято. Прийти к согласию — ваша задача, но помните, что она не всегда достижима. Кроме того, консенсус означает не единодушное согласие, но согласие большинства. Если после сбора информации и её анализа согласие все ещё не достигнуто (т.е. существуют разные мнения) абсолютно необходимо, чтобы менеджер проекта или один из его ведущих специалистов принял решение самостоятельно. Не откладывайте это и не проявляйте нерешительность — здесь нужна сильная рука лидера. Группе необходимо решение для продолжения работы, задержка с его принятием снижает мотивацию группы, и можно упустить благоприятный момент. Помните: лучше принять неверное решение, чем долго тянуть с его принятием и так никогда не узнать, что правильный выбор был совсем рядом.

Смена темпа работы

Несмотря на самые благие намерения и усилия по планированию, порой приходится увеличивать темп работы, чтобы уложиться в сроки или

отреагировать на изменившиеся условия. На первых порах сверхурочная работа требуется для решения большинства проектов; такая ситуация также типична для большинства проектов, которые нужно завершить в сжатые сроки, но бывают случаи, действительно требующие особой самотдачи, и речь здесь идёт не просто о паре лишних часов в неделю. Прежде чем принять решение об увеличении нагрузки на группу, нужно знать, когда это уместно и как делать это эффективно.

Когда нужно увеличить нагрузку

Менеджеру проекта и ведущим специалистам очень важно знать, когда нужно проситъ команду или её участника о дополнительных усилиях. Чтобы эта просьба нашла отклик, её причины должны быть обоснованы, а задачи — ясны каждому. Дополнительная работа потребуется, если необходимо следующее:

•

Завершить очередной этап в срок

Если очередной этап работы близится к концу и при этом есть ощущение, что можно не уложиться в сроки, нужно начинать сверхурочную работу уже сейчас, а не позже. Как сказано выше, не следует дожидаться начала следующего этапа, чтобы обнаружить проблемы с завершением текущего. Если чувствуете, что план может быть сорван, лучше приложить дополнительные усилия раньше, чем позже.

•

Наверстать упущениое

Если не удалось уложиться в срок при выполнении промежуточного этапа работы, упущенное придётся навёрстывать. Помимо этого, нужно завершить следующий этап в срок. Если вы всерьёз хотите вовремя закончить работу над проектом, приложите дополнительные усилия уже сейчас.

•

Справиться с возросшей конкуренцией

Если на рынке произошли серьёзные изменения, чтобы достигнуть успеха или просто не быть вытесненным конкурентами, реализацию проекта придётся ускорить. При этом, возможно, придётся добавлять новые функции к продукту или ужать план на месяц. Это потребует от всей группы полной самоотдачи.

Прежде чем просить кого бы то ни было посвятить проекту дополнительное время, убедитесь, что ваша просьба обоснованна. Постоянно прося команду об изменении графика работы или штурмуя каждую проблему работой в сверхурочное время, вы быстро исчерпаете продуктивность и эффективность команды. Нельзя заставлять людей слишком долго и слишком много работать сверхурочно. Этот подход следует использовать для навёрстывания упущенного или для адекватной реакции на внешние события, но нельзя делать сверхурочные частью нормального графика работы.

Как увеличивать нагрузку

Сверхурочная работа может потребоваться от одного из участников команды, функционального подразделения или от команды целиком. Не важно, сколько людей и кто именно привлекается к сверхурочной работе, существует ряд основных правил, которых необходимо придерживаться.

•

Определите продолжительность сверхурочных работ

Необходимо чётко определить продолжительность периода сверхурочной работы — он не может быть открытым или неопределённым. Следует поставить конечную задачу: уложиться в заданный срок или достичь определённого конечного результата;

•

Создайте комфорт

Сверхурочная работа должна доставлять удовольствие. Обязательно предлагайте питание работающим во время обеденного перерыва, а также тем, кто задерживается на работе допоздна. Следите за тем, чтобы работники могли взять из дома всё необходимое для работы. Организуйте максимально комфортабельную рабочую среду для тех, кто не может работать дома, а также организуйте для них беспрепятственный вход и выход из офиса по окончании обычного рабочего дня. Следует подумать об организации дополнительного обслуживания, например, привлечения посыльных для разных поручений или возможности сдать вещи в химчистку, чтобы помочь сотрудникам сохранить силы и сосредоточиться на работе.

•

Поддерживайте боевой дух

Когда от группы требуется самоотдача, проследите, чтобы все её участники работали сообща: все остаются, все помогают и вносят свой вклад. Если не нужно писать новый код, займите людей тестированием. Если нечего тестировать, бросьте всех на редактирование документации. Общая преданность делу и общая ответственность теснее сплотит группу и способствует развитию духа товарищества.

Из собственного опыта

В начале истории компании NuMega нам нередко приходилось работать сверхурочно. Мы часто работали по ночам и в выходные. Чтобы скрасить это время, дочь одного из сотрудников взялась готовить для нас. Она как раз училась стряпать и хотела попрактиковаться на нас. К счастью, у неё был талант и ей помогала мама. Конечно, свою пиццу мы получали тоже, но приготовленные ею обеды помогали нарушить однообразие.

Когда работы было особенно много, как правило, все работали сверхурочно, оставаясь после работы. Это было весёлое время, и часто сверхурочные часы превращались в вечеринку. Хотя людям приходилось жертвовать собственным временем и допоздна задерживаться на работе, все знали, для чего это делается, и верили, что наша работа важна для успеха общего дела. Когда работа была закончена, мы решили пригласить на празднование нового выпуска своих родственников. Мы понимали, какие неудобства сверхурочная работа доставила нашим семьям, и хотели как-то отблагодарить их за проявленное терпение.

- Информируйте о прогрессе

Если людям приходится много работать сверхурочно, они обязательно должны знать, что именно они являются двигателем проекта. Как менеджер проекта или ведущий специалист, вы должны показать команде, что их усилия оправдываются.

- Благодарите за труд

Это можно делать на собраниях или при личных встречах с отличившимися. Подумайте о специальных футболках, подарках и других наградах за особые заслуги. Непременно позаботьтесь, чтобы все работающие сверхурочно ощущали признательность за свой труд. Не забывайте и об их семьях. Все это отражается в первую очередь на них, и следует найти способ отблагодарить членов семей сотрудников за их жертвы.

Общие проблемы и решения

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Вы уверены, что завершили эту работу?

Вас никогда не спрашивали на контрольном собрании: «Вы уже закончили работу над X?» На самом деле это очень расплывчатый вопрос, однозначный ответ на который дать очень трудно. Означает ли это, что код написан и его можно скомпилировать? Или реализованная функция нормально работала пару раз, когда вы пытались использовать её? А, может быть, выполнено блочное тестирование программы на всех поддерживаемых платформах и конфигурациях? А что это означает для тестировщиков из соседнего отдела? Обязательно заведите для себя определение «законченной» работы, и ознакомьте с ним всех, иначе вы запросто обнаружите людей, в поте лица работающих над тем, что вы «закончили» несколько недель назад.

Борьба с нехваткой оборудования

Один из главных грехов фазы исполнения проекта — задержка работы из-за «нехватки оборудования». Если разработчику понадобится более ёмкий жёсткий диск, техническому писателю — новая мышь, а тестировщику — программа для мониторинга, следует доставить их немедленно. Никакие мелочи и пустяки не должны задерживать работу над проектами или снижать эффективность команды. Менеджер проекта должен неусыпно заботиться о личных нуждах, проблемах и потребностях каждого члена группы.

Из собственного опыта

В преддверии выпуска последней бета-версии одного из продуктов NuMega нам приходилось работать все ночи и выходные напролёт. К сожалению, нельзя сказать, что наш сервер был с нами солидарен: в течение дня он постоянно зависал без видимых причин, останавливая при этом всю сеть. В результате мы не могли проводить сдачу разработанных фрагментов ПО и делать контрольные сборки, что просто убивало нашу производительность.

Проблемы продолжались в течение двух недель — чтобы избавиться от них, мы перепробовали все возможные средства. Наконец, мы подумали, что сервер перегревается, так как он стоял в старом помещении в самом центре здания (мы ведь только начинали!). Возможно, там плохая вентиляция, думали мы. На следующий день мы купили большой вентилятор на стойке и пару дней обдували им сервер, но — увы! — он продолжал виснуть.

Наконец мы поумнели настолько, что просто купили новый сервер. Если бы мы догадались поменять его в первые два дня, то смогли бы сэкономить три недели работы, времени и усилий.

Навёрстывайте упущенное

Хотя мало кто будет спорить с тем, что выдерживать внутренние сроки очень важно, они не помогут составить план действий при угрозе срыва конечной даты. Если что-то пошло не так, нужно внести коррективы, чтобы вернуть проект на намеченный путь. Это ещё одна ситуация, когда менеджер проекта зарабатывает свой хлеб. Непременно контролируйте ход проекта по внутренним контрольным точкам, а если при исполнении промежуточных этапов не удаётся уложиться в срок, предпринимайте соответствующие действия.

Миритесь с недостатками своих сотрудников

Человек несовершенен. Не исключено, что придётся столкнуться с людьми, которые не желают верить в существование проблем или дают рекомендации на основе неполной информации. Спесь, как и застенчивость, утомление, истощение сил или личные проблемы могут стать причиной разных трудностей. Поддерживая работу над проектом в нужном русле, приходится иметь дело не только с технологиями, но и с человеческими недостатками. Изучите сильные и слабые черты характера каждого участника группы и обязательно руководствуйтесь этими сведениями, принимая решения.

Глава 13
Бета-тестирование

Бета-тестирование — это процесс проверки ПО внешними силами. В начале программы бета-тестирования новое ПО рассылается реальным или потенциальным заказчикам (бета-тестерам) для изучения, оценки и предоставления отзыва о его работе. Задача — получить от бета-тестеров объективную оценку возможностей и качества ПО.

Бета-тестирование позволяет получить ценные сведения о готовности ПО, прежде чем оно попадает к заказчикам. Программы бета-тестирования являются решающим фактором успеха в начале развития компаний, когда объём ресурсов, выделенных под проекты, ограничен. Бета-тестирование, как никакой другой способ, позволяет эффективно организовать тестирование программы. Оно не только расширяет возможности группы разработчиков в сфере контроля качества, но и обеспечивает поступление из внешнего мира объективных и достоверных отзывов о возможностях вашего ПО.

С другой стороны, плохо проведённое бета-тестирование означает, что не только разработчики, но и бета-тестеры лишь зря потратили на него своё время. В этой главе мы обсудим ключевые аспекты проведения хорошей программы бета-тестирования и способы улучшения программных продуктов с помощью бета-тестирования.

Ценность бета-тестирования

Прежде чем говорить о способах проведения хорошего бета-тестирования, обсудим, в чём вообще его польза. Не понимая ценности бета-тестирования или не веря в её существование, вы никогда не выделите достаточно времени и средств, чтобы провести его на должном уровне. Вот самые значительные аспекты пользы от бета-тестирования:

•

Проверка ПО в условиях реального мира

Независимо от того, насколько хорошо проведено внутреннее тестирование, воспроизвести в полном объёме все испытания, проводимые многочисленными бета-тестерами, было бы чрезвычайно сложно (если такая задача вообще выполнима). Если вы не ошиблись с подбором бета-тестеров, они помогут проверить работу новой программы на широком спектре вычислительных платформ и в самых разных ситуациях, все разнообразие которых вы скорее всего никогда не смогли бы охватить. Поскольку бета-тестирование выполняется реальными пользователями в реальных условиях, они часто обнаруживают такие ошибки, которые никогда бы не были найдены без их помощи.

Хорошие бета-тестеры позволяют проверить готовность ПО к использованию прежде, чем оно будет отправлено заказчику. Эта информация сама по себе уже стоит усилий, затраченных на проведение бета-тестирования;

•

Оценка работы ПО

Второй аспект пользы бета-тестирования — это получение отзывов о качестве функций, о производительности и о качестве пользовательского интерфейса ПО. Поскольку бета-тестеры работают с программой в самых разных условиях, они лучше всего смогут обеспечить вас информацией о пользовательских потребностях, симпатиях и антипатиях. Кроме того, они подскажут вам массу новых идей. Хотя на данном этапе вносить существенные изменения в программу не желательно, эти идеи послужат превосходной отправной точкой для работы над следующим выпуском;

•

Помощь в маркетинге

Бета-тестирование позволяет сделать программу более заметной на рынке и повысить доверие к ней, что будет вовсе не лишним для маркетинговой политики нового продукта. Бета-тестеры, получившие положительное впечатление от

работы с новой программой, — отличный источник материалов для пресс-релизов и рекламных каталогов. Они также пропагандируют вашу программу в данной отрасли, поскольку склонны обсуждать, рекомендовать и высказываться за использование программы как в своих фирмах, так и публично. Бета-тестеры также распространяют слухи о новой программе, что особенно ценно для её презентации и при выходе на рынок.

Однако от плохо организованной программы бета-тестирования вряд ли стоит ожидать помощи в маркетинге и раскрутке новой программы. Нужно тесно взаимодействовать с бета-тестерами, идти навстречу их нуждам и оказывать им всестороннюю поддержку. Кроме того, необходимо дать бета-тестерам почувствовать, что вместе с разработчиками они являются единой командой. Чем больше усилий вложено в бета-тестирование, тем больше шансов получить от него пользу.

•

Дополнительная рабочая сила

Один из главных аспектов бета-тестирования — возможность увеличить число работающих над проектом. При реализации как начальных, так и крупномасштабных проектов, программы бета-тестирования обеспечивают изрядное количество дополнительной рабочей силы, которая обошлась бы в сотни тысяч или даже миллионы долларов при найме по контракту или иным способом. В следующий раз, когда вы задумаетесь над вопросом, нужна ли вам программа бета-тестирования, проведите нехитрый расчёт. Умножьте число бета-тестеров на время, которое тратит каждый из них на испытания ПО, умножьте результат на размер почасовой оплаты труда наёмных тестировщиков и получите денежное выражение ценности бета-тестирования.

Самая распространённая ошибка при проведении бета-тестирования

В том, что результаты бета-тестирования становятся определяющими при формулировании основных требований к программе. Не следует использовать программу бета-тестирования для поиска функций, которые должны быть реализованы в программе, чтобы обеспечить её успех. Так подбирать функции уже слишком поздно, их нужно было определить на этапе формулирования требований и коммерческого анализа программы (см. главу 9). Если стало ясно, что программа обречена на провал на рынке, не старайтесь впихнуть новые функции в продукт, который вот-вот будет закончен. Возьмите тайм-аут и обсудите возможные альтернативы: может, лучше начать всё с самого начала, составив новый набор требований и план?

Помните: цель программы бета-тестирования — испытание и усовершенствование продукта, поиск идей на будущее и помощь в продвижении продукта на рынке. Во время работы с бета-версией идёт сбор отзывов о

реализации функций, возможных улучшениях, практичности и качестве программы. Кое-что из этого ещё можно изменить, но для добавления новых сложных функций этот период совершенно не годится.

Типы программ бета-тестирования

Программа бета-тестирования обычно состоит из нескольких фаз. Каждая последующая фаза включает в себя все большую группу пользователей, и результатом её является все более сложный и стабильный продукт. Несмотря на отсутствие формального определения фаз программы бета-тестирования, принятого в отрасли, приведённое ниже описание поможет составить общее представление о каждой фазе.

•

Фаза 1

К началу этой фазы должно быть написано 60-80% кода. Задача этой фазы — как можно скорее передать основные функции программы для испытаний лучшим бета-тестерам.

Допустим, первую фазу решено начать уже после завершения основных функции. В программе пока нет поддержки печати, реализована лишь малая часть справочной системы, нет сложных алгоритмов сортировки, фильтрации и функций пользовательского интерфейса. Однако продукт уже пригоден для работы, и отзывы окажут существенную помощь при тестировании и доводке его функций.

На этом этапе ещё есть время для внесения небольших, но важных изменений. Это вполне обычная практика, в ней нет ничего неожиданного. Необходим подходящий момент для улучшения ПО на основе отзывов клиентов и углублённого понимания продукта. Однако нужно быть уверенным, что изменения не нарушат плана и не снизят качества продукта. Если такой уверенности нет, подумайте, перевесит ли выгода от реализации новых функций затраты на расширение плана.

•

Фаза 2

К началу второй фазы код продукта готов на 100%. Все функции, которые намечено реализовать в окончательном выпуске продукта, запрограммированы и работают. Хотя существенно менять какие-либо функции не планируется, некоторые изменения всё же можно внести, если они действительно важны и не влекут за собой серьёзного риска. В этом случае тоже необходима уверенность в отсутствии негативного влияния изменений на исполнение плана проекта или качество продукта.

•

Фаза 3

В начале этой фазы функции продукта приобретают окончательный вид. Изменений возможностей программы или набора её функций не планируется. Согласно плану, внешний вид программы получит здесь окончательную форму, никаких важных изменений пользовательского интерфейса не ожидается. Набор функций блокируется, и группа концентрируется на повышении качества, производительности и «шлифовке», необходимой для успеха хорошей программы. На этом этапе задачей всей группы является подготовка продукта к коммерческому использованию путём тестирования, исправления ошибок и приведения документации к окончательному виду.

•

Маркетинговое бета-тестирование

Это особый тип программы бета-тестирования, в рамках которой потенциальные клиенты получают ПО, чтобы оценить, насколько оно соответствует их потребностям. Маркетинговое бета-тестирование особенно важно, когда новая программа даст клиентам существенные, потенциально революционные возможности, а также в случае продукта, имеющего большое значение для роста продаж компании. В таких ситуациях имеет смысл продемонстрировать клиентам успехи при создании продукта и спектр возможностей, которые они получат после его завершения.

Поскольку важно произвести положительное впечатление на клиентов, не спровоцировав у них необоснованных ожиданий, лучше использовать для маркетингового бета-тестирования одну из поздних бета-версий продукта (вторую или третью). Таким образом, можно быть уверенным, что клиенты увидят высококачественное и технически совершенное ПО.

Из собственного опыта

Когда работа в NuMega вступала взавершающие фазы бета-тестирования, мы всегда рассылали копии тем, кто «делает погоду» в отрасли. В этот список входили наши партнёры, основные производители, специализированные издания и аналитические службы, занятые в отрасли. Прежде всего мы хотели, чтобы ПО было свободно доступно каждому, кто может способствовать нашему успеху. Это было маркетинговое бета-тестирование в полном смысле этого слова, так как мы не ждали (и даже не хотели!) извещений о каких-либо ошибках или затруднениях,

Этот очень эффективный способ позволил донести наше имя и нашу продукцию до людей, способных к содействию в нашем деле.

Элементы программы бета-тестирования

Мы проанализируем ключевые элементы бета-тестирования и рассмотрим, что необходимо сделать для максимального повышения эффективности этого процесса в группе.

Начало программы бета-тестирования

Прежде чем приступать к набору первых бета-тестеров, надо решить ряд простых, но важных вопросов.

- Каковы основные черты вашего бета-тестера?

Определите профиль потенциального пользователя вашей программы. Какими навыками он должен обладать, какой опыт работы с подобными продуктами в данной отрасли ему нужен? Какие технологии, конфигурации и платформы должны быть ему доступны? Сколько времени и усилий потребуется от бета-тестеров на испытание программы?

- В чём основная цель программы бета-тестирования?

Независимо от того, начинаются ли испытания основного или неосновного выпуска ПО, нужно, чтобы люди сосредоточились на испытании ключевых функций программы, находящихся в разработке. Принципы работы этих функций должны быть ясны всем бета-тестерам, чтобы они смогли лучше проанализировать и оценить ПО.

- Сколько бета-тестеров потребуется?

Количество бета-тестеров критично для эффективности программы бета-тестирования. Если их слишком мало, то и информации будет собрано немного, а если их число чересчур велико, то можно не справиться с администрированием, управлением и поддержкой всех пользователей, что вызовет у бета-тестеров ощущение беспомощности и приведёт к потере важной информации. Как правило, лучше привлекать на 30% больше бета-тестеров, чем по расчётам понадобится для испытания программы. Это позволяет подстраховаться от

неизбежных накладок, возникающих, когда бета-тестеры неэффективно работают или вообще прекращают тестирование.

Обычно в первой фазе бета-тестирования задействовано меньше всего тестеров, так как продукт ещё не вполне готов для всесторонней оценки. К участию в первой фазе следует привлекать таких бета-тестеров, которые смогут справиться с проблемами и трудностями, а также помогут при отладке и диагностике ошибок. Однако во время третьей фазы к испытаниям программы должны присоединится все привлечённые для участия бета-тестеры.

•

Каковы сроки программ бета-тестирования?

Продолжительность программы бета-тестирования также имеет решающее значение для её эффективности. Хотя бета-тестирование, как правило, стремятся максимально сократить (чтобы быстрее отправить продукт заказчику), не следует урезать её настолько, чтобы лишиться преимуществ, которые она даёт.

Из собственного опыта

Поскольку в NuMega работают компетентные специалисты по технической поддержке и администратор бета-тестирования, мы могли проводить довольно большую программу бета-тестирования, приглашая до 200 бета-тестеров, которые помогали охватить широкий спектр приложений и платформ при испытаниях ПО. К тому же такое большое число участников бета-тестирования позволяло подстраховаться на случай отказа от участия или выхода из программы части бета-тестеров. Бета-тестеры играют очень важную роль в обеспечении качества выпускаемой нами продукции. Нам никогда бы не удалось воспроизвести все разнообразие окружений и конфигураций, используемых бета-тестерами, внутри компании, поэтому наша признательность бета-тестерам безгранична.

Продолжительность программы бета-тестирования зависит от вашей специализации, занимаемой на рынке ниши и сложности ПО. Потому не существует правил, применимых к любому случаю. Тем не менее мы обнаружили, что месяца более чем достаточно для проведения любой фазы бета-тестирования (в общей сложности вся программа занимает 3 месяца). За месяц бета-тестеры успевали поработать с продуктом и дать отзыв о нём. Первая фаза бета-тестирования особенно ценна тем, что позволяет вести испытания программы в параллели с разработкой. Вторая и третья фазы, посвящённые небольшим изменениям и исправлению ошибок, играют важную роль в повышении качества продукта по мере приближения срока начала поставок. Поскольку, начиная со

второй фазы, набор функций программы был «заморожен», оставалось целых два месяца на повышение качества, производительности и «шлифовку» программы.

Набор бета-тестеров

Первым этапом любой программы бета-тестирования является поиск и привлечение бета-тестеров. Один из самых важных критериев для набора бета-тестеров — наличие у них личной заинтересованности в использовании и успехе ПО. При отборе наилучших бета-тестеров надо уделить особое внимание их мотивации. Вот наилучшие источники бета-тестеров:

•
Потребители (внешние заинтересованные лица, внутренние конечные пользователи)

Чаще всего именно потребители больше всех заинтересованы в разработке нового продукта и поэтому могут стать прекрасными бета-тестерами. У них имеется законное желание наблюдать за разработкой продукта в соответствии с их потребностями и в их собственном окружении. Даже в отсутствие настоящих клиентов у вас наверняка найдётся несколько потенциальных клиентов, серьёзно заинтересованных в создаваемом вами ПО.

•
Персонал технической поддержки

Другой замечательный источник бета-тестеров — группа технической поддержки. Клиенты часто звонят в службу технической поддержки, чтобы пожаловаться на ошибки или спросить, как работает та или иная функция программы. Если в бета-версии программы проблемы клиента будут решены (путём исправления ошибки или облегчения работы с программой), он также может быть заинтересован в том, чтобы стать бета-тестером программы.

•
Посетители Web-узла компании
Как правило, посетители корпоративного Web-узла интересуются продуктами компании, поэтому им также может быть интересно поучаствовать в бета-тестировании. Обязательно дайте эту возможность потенциальным пользователям, зарегистрировавшимся через Web. Этот способ позволяет существенно расширить штат бета-тестеров, однако надо убедиться в наличии у них достаточной для этого квалификации.

•

Партнёры и союзники

Партнёры и поставщики компании также могут стать отличными бета-тестерами. При наличии деловых или технических связей с другими компаниями попробуйте привлечь их к бета-тестированию продукта. Взаимный интерес может быть достаточно велик, чтобы протестировать совместную работу продуктов.

Необходимо составить профили бета-тестеров, которых хотелось бы привлечь к участию в программе бета-тестирования. Например, желательно, чтобы часть бета-тестеров пользовалась локализованными версиями Microsoft Windows для тестирования продукта, предназначенного для японского, корейского и китайского рынков, а другая часть бета-тестеров работала на многопроцессорных системах. Не важно, какова формулировка задач, главное, чтобы при распределении их между бета-тестерами последние обеспечивали доскональную проверку ПО в соответствии с требованиями программы бета-тестирования.

Кроме того, перед получением ПО каждый бета-тестер должен подписать соглашение о неразглашении коммерческой тайны. Это соглашение поможет обеспечить конфиденциальность ПО и свести к минимуму потенциальную утечку информации к конкурентам.

Взаимодействие с бета-тестерами

Регулярный обмен с бета-тестерами достоверной и правдивой информацией имеет решающее значение для успеха программы бета-тестирования. Нужно заранее определить все ожидания и обеспечить своевременное решение любых проблем, а также удовлетворение любых потенциальных потребностей и запросов. Вот основные правила.

•

Определите ожидания

Чётко обозначьте начало и конец бета-тестирования, проинформируйте бета-тестеров о новшествах в текущем выпуске ПО, заострив внимание на всех специфичных моментах, требующих проверки. Поставьте тестеров в известность о том, что желательно как можно скорее сообщать о найденных ошибках, а после завершении бета-тестирования — заполнить анкету. Дайте понять, что их отзывы представляют большую ценность и вы постараетесь как можно скорее устранить найденные неполадки.

•

О серьёзных проблемах и найденных решениях должны знать все

Если какая-то важная часть программы не работает, дайте бета-тестерам знать о возникшей проблеме и что над её решением уже работают. Для этой цели лучше всего подходит электронная почта. Однако если требуется более

интерактивное обсуждение трудностей, проблем и потребностей, надо подумать об открытии на вашем Web-узле специализированной телеконференции.

- Решайте проблемы по мере их возникновения

Бета-тестеры должны сообщать о своих трудностях и обнаруженных проблемах специализированному (и достаточно опытному) персоналу из группы поддержки. Если у вас пока нет группы поддержки, нужно выделить для работы с бета-тестерами одного из разработчиков. В любом случае нужно регистрировать возникшие затруднения и обнаруженные ошибки.

Также следует рассмотреть возможность создания или, если надо, покупки системы для обработки поступающих сообщений об ошибках, обнаруженных в бета-версии. Системы по обработке сообщений специально сконструированы для обработки обращений клиентов, желающих сообщить о возникших трудностях. Возможность следить за сообщениями, поступающими от бета-тестеров, регистрировать их адреса и контактную информацию, обстоятельства возникновения ошибки, частоту обращений и другие данные, а также взаимодействовать с группой технической поддержки представляет настоящую ценность. Эта задача превосходно решается с помощью систем обработки поступающих сообщений, в то время как большинство систем отслеживания ошибок с ней не справляется. Присланное пользователем сообщение об ошибке следует регистрировать в системе, только если оно достоверно и ошибка воспроизводится.

- Доводите решение проблемы до конца

Как правило, лучше выделить специального разработчика, который в прямом контакте с бета-тестером доведёт решение найденных проблем до конца. Прямое взаимодействие ускоряет процесс разрешения проблем и способствует упрочению связей между бета-тестерами и группой разработчиков. Кроме того, это прекрасная возможность для технических специалистов поближе познакомиться с клиентами, их проблемами и их манерой работы с программой.

- Не закрывайте глаза на проблемы бета-тестеров

Вам наверняка придётся столкнуться с проблемами, возникающими у бета-тестеров. Ваше ПО может повредить реестр их ОС, а программа удаления ПО может заодно стереть ключевые системные файлы. Обязательно проследите, чтобы под рукой всегда были нужные люди, способные разобраться с этими проблемами.

Оценка прогресса бета-тестирования

Чтобы облегчить начало программы бета-тестирования, бета-тестеры должны быстро получить и установить ПО. Это можно сделать, опубликовав ПО в Web или разослав его на CD-ROM. Если выбрать публикацию ПО в Web, можно отслеживать число скачиваний и сравнивать его с числом привлечённых бета-тестеров. С другой стороны, рассылка бета-версий на CD-ROM позволяет быть уверенным, что тестеры получат продукт целиком.

Когда ПО будет у каждого, проследите, чтобы все бета-тестеры работали с ним. В любом случае разумно обязать администратора бета-тестирования обзвонить или списаться по электронной почте с каждым бета-тестером, чтобы убедиться, что все они работают с ПО. Важно стимулировать усилия бета-тестеров и не дать им остановиться на полпути.

В начале поступления отзывов выделите самые распространённые трудности, с которыми пришлось столкнуться бета-тестерам. В частности, немедленного решения требуют проблемы, возникшие из-за ошибок при установке или из-за недостаточного качества, ставших причиной затруднений у значительной части бета-тестеров. Самыми подходящими кандидатами на эту роль являются администратор бета-тестирования и ведущий специалист технической поддержки, реагирующие на любые возникающие проблемы. Такие неполадки нужно устранить как можно скорее, иначе под угрозой окажется вся программа бета-тестирования и её польза для группы.

Завершение программы бета-тестирования

При свёртывании бега-тестирования обязательно попросите прислать сведения обо всех ошибках и трудностях, до сих пор оставшихся без внимания. Это также самое подходящее время, чтобы разослать бета-тестерам анкету, представляющую формальный план отзыва о работе продукта. С её помощью очень легко выяснить, насколько интенсивно и для каких целей использовался продукт, какие при этом возникли трудности и проблемы. Такая информация слишком часто остаётся в голове бета-тестера, а анкета — прекрасный способ выудить оттуда все ценные мысли и идеи, чтобы позже извлечь из них выгоду. Ниже приводятся примеры вопросов анкеты.

• Как долго использовался продукт?

• Удалось ли легко и быстро установить его?

• Помог ли продукт быстро решить ваши проблемы и достичь поставленных целей?

• Какая из функций программы оказалась самой полезной для вас?

• Какая из функций оказалась наименее полезной и почему?

• Какую из функций вам больше всего хотелось бы увидеть в следующем выпуске программы?

• Как можно было бы улучшить документацию или справочную систему ПО?

• Оправдала ли производительность продукта ваши ожидания?

• Собираетесь ли вы регулярно использовать продукт?

• Собираетесь ли вы стимулировать использование продукта в вашей группе? Почему?

• Порекомендовали бы вы его другим? Почему?

• Готов ли продукт? Если нет, то почему?

Обязательно поинтересуйтесь, нет ли у бета-тестеров комментариев для пресс-релизов. Оказалось, что авторами лучших отзывов о продуктах компании NuMega были именно участники бета-тестирования. Мы донесли эти отзывы до всех работников компании и клиентов и, конечно же, опубликовали.

Из собственного опыта

По окончании бета-тестирования мы в NuMega оцениваем эффективность каждого бета-тестера как высокую, среднюю или низкую. Лучшим бета-тестерам мы посылаем бесплатные программы, фирменные футболки и даже куртки. Средним достаётся только один подарок, а тех, кто не принимал заметного участия в тестировании, мы вовсе исключаем из программы. Со временем мы почувствовали, что можем положиться на наших бета-тестеров как в получении объективных отзывов, так и в испытании новых продуктов при работе над бета-версиями и кандидатами на выпуск.

Как-то раз мы выпускали бета-версию в пятницу около четырёх часов дня. Разослав бета-тестерам сообщения, мы уже готовились отправиться по домам, когда один из тестеров прислал ответ, где говорил, что отложил свидание лишь для того, чтобы увидеть наш новый выпуск! В тот же вечер он сообщил, что на первый взгляд новый выпуск неплох, но он ещё не успел полностью его изучить. Вот так порой сталкиваются преданность делу и личные дела!

Поощрение лучших бета-тестеров

По завершении бета-тестирования важно проанализировать, что прошло хорошо, а что — нет. В рамках этого анализа нужно оценить работу самих бета-тестеров. Необходимо выделить и поощрить тех, кто вовремя предоставлял ценную информацию. Если какие-либо тестеры вообще не предоставили никакой информации, от их услуг лучше отказаться, чтобы освободить место для новых участников.

Менеджер бета-тестирования

Курирует бета-тестирование и управляет её исполнением. Поскольку программы бета-тестирования зачастую масштабны и сложны, эту работу должен выполнять штатный специалист. Это может показаться чересчур для такой

работы, но с учётом ценности хорошо проведённого бета-тестирования, все затраты должны оправдаться с лихвой. В компетенцию менеджера бета-тестирования входит следующее:

- Определение целей и задач

Менеджер бета-тестирования должен следить за тем, что для программы бета-тестирования определены цели, профиль клиента, масштаб и продолжительность.

- Набор бета-тестеров

Менеджер бета-тестирования должен следить, чтобы к моменту выпуска бета-версии было набрано достаточно бета-тестеров, готовых принять участие в испытаниях программы. Очевидно, что любая задержка с набором достаточного числа бета-тестеров уменьшит объём и снизит качество получаемой информации.

- Распространение ПО

Менеджер бета-тестирования составляет и реализует план распространения ПО среди бета-тестеров как через Интернет, так и на физических носителях. Он должен обеспечить быструю и эффективную передачу ПО каждому бета-тестеру. Если решено распространять программы через сеть, надо убедиться в наличии линий связи и серверов, необходимых для поддержки большого числа одновременных обращений бета-тестеров. Если для распространения ПО выбран физический носитель, необходимо обеспечить достаточные производственные мощности для своевременного распространения ПО.

- Распространение сведений о состоянии бета-тестирования

Менеджер является информационным центром, снабжающим всех бета-тестеров сведениями о состоянии программы бета-тестирования. В частности, он объявляет начало и конец бета-тестирования, а также сообщает о новом доступном ПО. Он также является связующим звеном между бета-тестерами и группой создателей продукта, обеспечивая обмен важными сведениями, включая изменения в плане программы бета-тестирования, появление обновлений ПО, исправлений критических ошибок и передачу специальных запросов на отзывы.

- Управление результатами бета-тестирования

От менеджера во многом зависит успех программы бета-тестирования. Почти сразу после её начала он должен обзвонить бета-тестеров, чтобы проверить, получили ли они ПО и, если да, начали ли работать с ним. Менеджер бета-тестирования должен следить за успехами каждого бета-тестера и обязательно стимулировать отстающих, чтобы они установили ПО и приступили к работе.

•

Завершение программы бета-тестирования

Менеджер должен так провести свёртывание бета-тестирования, чтобы не разочаровать тестеров, ещё не закончивших работу с продуктом. Менеджер запрашивает у тестеров их окончательные идеи, комментарии и сообщения об ошибках, а также обеспечивает их своевременное получение.

•

Поощрение бета-тестеров

Бета-тестеры — чрезвычайно ценные участники проекта. Хороших тестеров необходимо поощрять, плохих — исключать из программы.

Общие проблемы и решения

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Начинайте пораньше

Набор бета-тестеров требует времени, как и подписание соглашения о неразглашении коммерческой тайны. Набор непременно нужно начинать задолго до готовности бета-версии программы. Для программы, в которой участвует 200 бета-тестеров, я рекомендую оставлять 60-90 дней.

Не забывайте, что бета-тестеры заняты своими делами не меньше вас и могут приступить к оценке ПО с опозданием. В общем случае эту проблему можно решить тремя способами. Во-первых, если продукт сложен и для его установки нужны значительные ресурсы, можно послать к бета-тестерам своих людей, чтобы они помогли установить и запустить ПО для тестирования. Во-вторых, можно предположить, что до трети бета-тестеров не смогут начать тестирование вовремя или вообще откажутся от него. Поэтому следует включить в программу дополнительных бета-тестеров, чтобы компенсировать их возможный недостаток и не допустить задержки. В-третьих, следует непременно связываться с каждым бета-тестером лично (если можно, по телефону), чтобы следить за их успехами в начале проекта и составить представление об их возможности участвовать в программе бета-тестирования.

Бета-версии должны быть проверены

Как я говорил в главе 1, выпуск каждой бета-версии знаменует завершение крупного промежуточного этапа в работе над проектом. Таким образом, прежде чем рассылать бета-версию бета-тестерам, нужно завершить период стабилизации и интеграции. Этот период (обычно 1-2 недели) используется для тестирования, исправления ошибок и решения серьёзных проблем. Качество ПО должно быть достаточным для работы на местах бета-тестирования. До выпуска продукта во внешний мир надо подумать о проведении внутренних испытаний, о которых пойдёт речь в главе 14. Бета-версия — это ещё не окончательная версия ПО, однако изложенные в этой главе концепции помогут извлечь пользу даже из неё.

Необходима мощная инфраструктура

Для нормального проведения бета-тестирования нужна подходящая программная и аппаратная инфраструктура. Скажем, для управления поступающей от бета-тестеров информацией, статусом соглашений о неразглашении коммерческой тайны, оценки работы бета-тестеров скорее всего потребуется специальная программа. Нельзя недооценивать объём информации, с которым придётся работать.

Как справиться с потоком информации

Самой большую пользу от бета-тестирования представляет информация, которую оно позволяет получить. Процедуры передачи, обработки и продвижения информации должны быть чёткими и понятными. Для поддержки бета-тестирования обязательно нужны компетентные и хорошо обученные специалисты. Также необходимо оставаться в контакте с бета-тестерами в течение всего процесса бета-тестирования.

Не жалейте времени

Для бета-тестирования должно быть выделено достаточно времени. Начав его в понедельник, не надейтесь закончить его в пятницу, если рассчитываете на хорошие результаты.

Собирайте отзывы

Как я уже говорил, ценность программы бета-тестирования определяется числом отзывов, которые она помогает собрать. Если отзывов слишком мало, нужно собрать их. Звоните и спрашивайте, установили ли тестеры программу и успешно ли они с ней работают. Пишите по электронной почте, чтобы убедиться, что у всех бета-тестеров всё идёт нормально. Анкеты также помогают собрать сведения, позволяющие решать проблемы и следить, чтобы реализация проекта шла по намеченному пути.

Глава 14
Кандидат на выпуск

Вот и исправлена последняя ошибка — всё готово к окончательной сборке ПО, которая станет «кандидатом на выпуск, (relise candidate, RC). Хочется думать,

что на этом этапе неприятностей уж точно не случится, однако вероятность возникновения серьёзных проблем все ещё велика. В конце концов, после выпуска с вашим ПО будут работать сотни, тысячи и даже миллионы пользователей. Можно ли быть заранее уверенным в готовности продукта? Как знать наверняка, что последний набор изменений не привёл к существенному падению производительности или что функцию, протестированную на прошлой неделе, не нарушили вчера или позавчера? Нельзя просто сидеть и надеяться, что всё идёт хорошо. Отзыв ПО из производства или из сети после того, как было публично объявлено о его выходе, чреват не только большими убытками, но и потерей репутации компании.

При работе над кандидатом на выпуск проводится систематическая и объективная проверка окончательной сборки программного продукта, чтобы выяснить, готова ли она к выходу. В этой главе будут раскрыты базовые принципы организации работы над кандидатами на выпуск, преследующей цель подготовки ПО к выходу во внешний мир.

Начальные требования

К началу тестирования кандидата на выпуск все работы над продуктом (кроме собственно испытаний кандидата) должны быть закончены. Несмотря на это простое требование, всегда есть сильное искушение найти ещё несколько ошибок или внести изменения в программу и её документацию. Начав работу над кандидатом на выпуск, следует вести очень строгий контроль любых изменений. Не обманывайте себя, думая, что всё готово, когда на самом деле все наоборот. Чтобы внести ясность в этот вопрос, пройдёмся по основным требованиям, предъявляемым к кандидатам.

- Готовы все функции программы

Все без исключения функции должны быть завершены на 100%. Участники команды должны быть уверены, что цель разработки ПО достигнута и в случае успешного завершения тестирования в ПО больше не планируется вносить никаких изменений.

- Справочные материалы приведены в окончательный вид

Команда полностью завершила работу над справочной системой программы, электронной документацией, информационными файлами и электронными учебниками. Материалы проанализированы, выверены и полностью закончены. Можно дать ещё неделю на завершение подготовки

печатной документации, но электронная документация, которая будет поставляться с ПО, должна быть готова.

•

Завершена последняя проверка пользовательского интерфейса

Группа уже закончила оценку и доводку пользовательского интерфейса, так что интерфейс останется неизменным вплоть до отправки продукта заказчику;

•

Закончено тестирование программы

Группа выполнила план тестирования в полном объёме: проведено блочное, системное, нагрузочное тестирование, тестирование производительности и испытание пользовательского интерфейса, а также автоматизированное тестирование. Все тесты пройдены, по крайней мере, известны все неполадки и решено поставлять продукт, не устраняя их.

•

Все ошибки исправлены

Все ошибки, которые планировалось исправить, уже исправлены. Что касается остальных ошибок, то, проанализировав все сообщения о неисправленных ошибках, группа пришла к выводу, что они не могут или не должны быть исправлены в этом выпуске ПО.

Тестирование кандидата на выпуск

Фактически кандидат на выпуск и есть та версия ПО, которая будет отправлена заказчику, если последний цикл тёстирования не выявит серьёзных проблем. Даже если время ограничено, всё равно нужно протестировать ключевые функции ПО на его окончательной сборке, а это значит, что тестирование придётся вести очень напористо. Рассмотрим основные процедуры тестирования кандидатов на выпуск.

Создание окончательной сборки

Обеспечив соответствие программы начальным требованиям к кандидатам на выпуск, можно приступать к созданию окончательной сборки ПО. На этом этапе необходимо:

• остановить все изменения в системе управления версиями и заблокировать систему управления исходным текстом;

• создать одну полную сборку программы на основе окончательной версии исходного текста (с использованием оптимизации при компиляции);

• прекратить создание новых сборок — с этого дня ежедневная сборка ПО отменяется;

• пометить нужные файлы в системе управления исходным текстом;

• уведомить всех участников команды о том, что кандидат на выпуск готов!

Автоматизированное и ручное тестирование

Одной из трудностей в работе с кандидатами на выпуск является отбор функций, которые должны быть испытаны в окончательной сборке. Помните: полностью протестировать весь продукт ещё раз не получится, так как на это уйдет несколько месяцев (а то и лет). Вместо этого нужно составить конкретный и чётко сформулированный план тестирования кандидата на выпуск, который можно будет выполнить в очень сжатые сроки. При этом ваши вложения в автоматизацию тестирования воздадутся сторицей. Если в своё время тестирование ключевых функций продукта было автоматизировано, затруднений возникнуть не должно. Тестирование избранных функций кандидата на выпуск должно быть на 70-80 или даже на 90% автоматизировано, что позволяет максимально сократить период испытаний кандидата на выпуск. При отсутствии полной автоматизации потребуется дополнительное время на проведение тестирования вручную. Помимо исполнения набора автоматизированных тестов, при проведении ручных тестов следует сосредоточиться на:

• проверке установки и функций, связанных с лицензированием;

• тестировании базовой функциональности продукта, а именно:

— ключевых функций;

— основных элементов пользовательского интерфейса;

— важнейших ссылок в справочной системе;

— программ-примеров и электронных учебников;

• самых необходимых тестах производительности и нагрузочном тестировании;

• других специфичных для данного проекта функциях, которые требуется протестировать.

Все эти тесты нужно проводить внутри группы. Однако, поскольку скоро ПО будет отправлено заказчикам, надо создать небольшую группу из тщательно отобранных бета-тестеров. Они смогут оценить пригодность программы для коммерческого использования. С помощью такого рода внешней проверки можно получить объективные данные о готовности ПО к выпуску. В последние дни работы над любой программой на плечи ведущих участников программы ложится огромная нагрузка, и сведения из внешнего мира могут дать им непредвзятую оценку ПО, необходимую для принятия правильных решений. В конце концов, если до отправки ПО заказчику ни один из десятка пользователей не смог добиться успеха в работе с программой, как можно ожидать, что у сотен и тысяч других пользователей, которые получат программу, всё получится?

Отбор части лучших клиентов для участия в испытаниях кандидата на выпуск находится в компетенции администратора бета-тестирования. Хорошие тестировщики должны согласиться поработать с ПО в реальных ситуациях, быть способными уложиться в сроки плана работы с кандидатом и предоставить информацию о неполадках в случае их возникновения. Во время работы над

кандидатом на выпуск нужно практически ежедневно контактировать с тестировщиками, чтобы контролировать их работу и собирать как положительные, так и отрицательные отзывы. Поскольку с тестировщиками кандидатов на выпуск часто складываются даже более интенсивные взаимоотношения, чем с другими бета-тестерами, их обычно меньше, чем других бета-тестеров.

Из собственного опыта

Для работы над кандидатами на выпуск в NuMega обычно приглашают только лучших бета-тестеров. По завершении внутренних тестов мы посылаем программу тестерам кандидатов на выпуск с просьбой вынести в течение 3-5 дней вердикт: готово ПО или нет. Администратор бета-тестирования остаётся на связи с тестерами: если возникают проблемы, тестеры получают приоритетную поддержку, часто прямо от разработчиков. Когда сильно поджимают сроки, отзывы клиентов могут быть как весьма обнадёживающими, так и очень тревожными. После начала поставок продукта мы часто дарим нашим тестерам его копию вместе с футболками разработчиков в благодарность за их труд.

Обеспечьте «мягкую посадку» проекта

Возвращаясь к аналогии с самолётом (см. главу 12), можно сказать, что задача в том, чтобы в сжатые сроки постепенно вывести проект «на снижение» и плавно «посадить» его. Как и экипажу самолёта, вам нужен набор предопределённых процедур, направляющих действия при «посадке». Вот и пришла пора убрать столики и привести спинки кресел в вертикальное положение.

Поскольку работа над кандидатом на выпуск — критический период проекта, он требует открытого обмена точной информацией о состоянии проекта, отражающей как успехи, так и неудачи. В этот период необходимо оперативно принимать решения, чтобы устранять ключевые затруднения, исправлять ошибки, контролировать риск и выбирать между альтернативами. Чтобы справиться с этими задачами, нужно создать «штурмовую группу», состоящую из лидеров всех функциональных подразделений:

- разработчиков;
- тестировщиков;
- группы по обучению пользователей;
- группы инженерной психологии;
- технологов;
- технической поддержки;
- менеджера продукта;

• администратора бета-тестирования.

Из собственного опыта

Пока в NuMega не начала работать «штурмовая группа», на завершающих стадиях работы над проектами всегда возникала куча проблем. Они были буквально везде: с мониторингом текущего состояния работы над программой, передачей сообщений о найденных ошибках, назначением ответственных за их устранение, координацией тестирования, контролем слухов, принятием решений и обменом информацией в группе. Когда численность групп возросла и от создания отдельных продуктов мы перешли к разработке пакетов программ, наши проблемы особенно обострились. Порой это напоминало шоу трёх простофиль

1

. К счастью, мы всё же догадались создать «штурмовую группу», что позволило решать большинство проблем, преследовавших нас на завершающих стадиях проекта.

Идея состоит в создании штабного помещения — единственного места, где можно ставить, анализировать и обсуждать проблемы. Антикризисные собрания должны проводится ежедневно в одной и той же комнате. Такой подход, использующий предварительное планирование, позволяет формализовать и упорядочить область ответственности каждого специалиста, докладывающего о состоянии дел в ней, а также о накопившихся проблемах. По мере проведения заключительных тестов и поступления клиентских отзывов извне, накапливается значительная информация, которую можно и нужно довести до сведения каждого члена группы. Даже когда проблемы и трудности отсутствуют, всё равно неплохо собраться всем вместе, чтобы разделить эту хорошую новость. Наконец, если требуется немедленно принять критически важное решение, можно просто созвать экстренное собрание «штурмовой группы».

Если что-то идёт не так, стоит задуматься

При проведении последних тестов служба поддержки, тестировщики, администратор бета-тестирования, инженеры да и любой участник группы могут обнаружить серьёзную проблему. Её следует рассмотреть на собрании «штурмовой группы», которая может предпринять следующие действия:

•

Прояснить проблему

Прежде всего надо убедиться в реальности проблемы, затем исследовать её природу и определить её значение для проекта: выяснить, воспроизводится ли она, а также какие и сколько платформ она затрагивает.

•

Оценить затраты на исправление ошибок или внесение изменений

Сначала нужно определить, можно ли устранить неполадку в принципе, а затем оценить масштаб и величину изменений, которые для этого придётся внести в программу.

•

Решить, делать ли новую сборку программы

Следует взвесить затраты на решение проблемы и сравнить их с ущербом, которая она нанесёт, если оставить её без решения. Достаточно ли серьёзна проблема, чтобы оправдать затраченное на её решение время, особенно если при этом задержится выпуск продукта?

Если решено создать новую сборку, следует назвать её с учётом схемы именования RC

n

+1, где

n

— номер версии последнего кандидата на выпуск. Проследите, чтобы номер сборки нового кандидата стал известен каждому.

•

Выполнить повторный цикл тестирования кандидата на выпуск

Если неполадка локальна, достаточно повторного тестирования лишь той части программы, что была изменена при её устранении. Однако повторное тестирование программы установки следует провести в любом случае.

Эти решения очень важны, и следует проследить, чтобы их принимали компетентные представители каждого из функциональных подразделений. Поскольку эти проблемы решаются на собраниях «штурмовой группы» с участием всех ключевых специалистов, можно быть уверенным в наличии достоверных данных, солидном опыте принимающих решение и в распространении сведений о принятых решениях от первого лица.

Если всё в порядке, можно заканчивать

Если тестирование кандидата на выпуск успешно прошло в полном объёме, его можно утвердить. Последнее утверждение продукта — во многом формальность, так как именно успешное окончание испытаний кандидата определяет момент, когда проект можно считать завершённым. Однако эта

процедура гарантирует, что каждый внёсший свой вклад в создание проекта, будет готов поддержать проект и, если понадобится, отстаивать его. Проект должны утвердить следующие специалисты.

- Технические специалисты

Все подразделения: разработчики, тестировщики, специалисты по обучению пользователей, инженерные психологи и технологи — должны единогласно утвердить проект. Это означает, что каждое подразделение внесло свой вклад в создание реального продукта и готово дать ему «зелёный свет». В рамках модели, принятой в NuMega, за готовность проекта в конечном счёте отвечает менеджер проекта, а сама готовность определяется по согласованию с командой. Технические специалисты первыми ставят свою подпись под проектом, без их визы проект дальше не пойдёт.

- Менеджеры продукта

Если группа менеджеров утвердила проект, это означает, что качество реализации функций ПО позволяет выйти с ним на рынок. Планы лицензирования, формирования цены, обучения продавцов, производства дистрибутивов и сбыта также уже составлены, и можно приступать к их реализации.

- Группа технической поддержки

Виза группы технической поддержки означает, что ни одна критическая проблема не осталась нерешённой и группа готова осуществлять поддержку продукта после выхода его на рынок.

- Заказчики

Виза заказчиков означает, что продукт готов к использованию в их производственном или коммерческом окружении.

Когда продукт готов, можно передать его заказчику

После того, как все дали «зелёный свет» можно передавать проект заказчику и принимать поздравления с окончанием работы! Но прежде, чем считать проект завершённым, обязательно прочитайте следующую главу, «Закрытие проекта» (вот тогда проект действительно будет закрыт).

Общие проблемы и решения

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Отсутствие руководства

Одна из наиболее распространённых проблем, мешающих плавно завершить проект, — нехватка руководства и дисциплины на завершающих стадиях. В этот период менеджер проекта отрабатывает свой оклад, поскольку должен быть кто-то один, ответственный за выпуск ПО. Обеспечить эффективную работу на завершающих этапах проекта нелегко. Возможно, придётся задержать группу в выходные, чтобы провести повторные тесты или отвергнуть последний набор действительно классных изменений из-за слишком высокого риска. Не ограничивайте себя в средствах. чтобы выполнить эту работу в лучшем виде.

Обмен информацией

Когда на группу обрушиваются все заботы, связанные с выходом ПО, обмен информацией между участниками группы часто нарушается, что провоцирует возникновение слухов и сомнительных разговоров. Обязательно нужно организовать управление работой над проектом вплоть до его закрытия, обмен информацией о его состоянии и решение любых проблем по мере их возникновения.

Ответственность

Поскольку в производстве ПО участвует много людей и постоянно приходится принимать решения, важно назначить ответственных в каждом подразделении, работающем над кандидатом на выпуск. Очень плохо, когда ответственность постоянно переходит от одних людей к другим. Нужны люди, которые могут отвечать за вверенные им команды и за принятые решения.

План тестирования

Необходим конкретный план тестирования кандидата на выпуск. Нельзя тестировать все подряд, на это просто не хватит времени. Нужно сосредоточиться на важнейших фрагментах. Однако у многих групп отсутствует чёткое представление о том, что и как нужно делать. Заключительное тестирование должно быть продумано, прежде чем проект вступит в фазу работы с кандидатом на выпуск.

Автоматизация

Если тестирование автоматизировано недостаточно, дело плохо. Возможность автоматизированного тестирования ключевых функций на окончательной сборке программы (и на новых сборках программы, если они будут созданы) имеет решающее значение для своевременной и полной проверки продукта. Нельзя тратить время на тестирование каждого кандидата на выпуск вручную.

Глава 15
Закрытие проекта

Наконец, проект завершён, программа отправлена заказчикам! Это все, не так ли? Ещё нет. Завершить работу над проектом — это больше, чем просто вынести его за дверь и уйти домой. В проект вложено много времени и сил, команде пришлось принести большие жертвы, теперь критически важно должным образом провести закрытие проекта, не забыв никого из участников. Если отказаться от этого этапа, можно упустить прекрасную возможность отблагодарить людей, выразить команде признательность за внесённый вклад и подготовить её к работе над следующим проектом.

Почему это так важно?

Закрытие проекта венчает работу команды. Оно также позволяет участникам осознать всю важность проекта, а также почувствовать, что их вклад и самопожертвование получили признание и были оценены по достоинству. Слишком часто упадок сил после завершения проекта ведёт к неразберихе и даже к депрессии. Люди должны быть уверены, что их усилия, сверхурочные часы и бессонные ночи не пропали зря. Они могут спросить себя: «Была ли моя работа замечена? Стоило ли так на напрягаться?» — или, что ещё важнее: «Буду ли я выкладываться в следующий раз?» Правильно проведённое закрытие проекта даёт ответ на эти вопросы. При правильно проведённом закрытии необходимо:

- известить всех об окончании проекта и о передаче программы заказчику;
- выделить индивидуальные достижения, вклад и преданность делу;
- отметить общие усилия и эффективность работы команды в целом;
- помочь участникам команды увидеть их ошибки и извлечь из них урок;
- решить текущие проблемы с кадрами или проектом;
- начать подготовку к следующему проекту.

Как это делается?

Провести закрытие проекта несложно, но для этого нужно решить множество задач.

Передача программы

Выпуск во внешний мир программного продукта его создателями должен быть общественным событием. Нельзя придумать более удачной церемонии закрытия проекта, чем передача прав собственности командой заказчику. Это событие должно происходить на глазах у всей команды. Оно представляет собой яркий и запоминающийся момент для каждого участника работы над проектом.

Например, если ПО передаётся производственной организации, можно устроить небольшую церемонию, во время которой менеджер проекта передаёт компакт-диски с ПО представителю этой организации. Если программа передаётся через сеть, то поводом для сбора всех участников команды может быть последнее нажатие клавиши ввода. Как бы это ни происходило, главное — собрать всю команду, чтобы каждый участник узнал о передаче проекта заказчику.

Конечно, событию должна предшествовать краткая речь. Всегда весело, когда люди вспоминают самые трудные задачи проекта и как команда решала их сообща. Забавно припомнить самые комичные эпизоды работы над проектом, поинтересовавшись, как же всё-таки удалось пережить их живым и невредимым!

А теперь рассмотрим противоположную ситуацию, когда закрытие не проводилось. Если участникам команды приходится спрашивать, отправили ли проект заказчику или они обнаруживают, что их не пригласили на передачу проекта, они ощущают себя вовсе не игроками одной команды, а скорее винтиками, которые сослужили свою службу в составе большой машины, но так и не стали её неотъемлемыми частями. Так быть не должно.

Заключительное письмо

Если команда многочисленна или её участники разбросаны по обширной территории, собрать всех вместе может быть сложно. В таких случаях обычно прибегают к рассылке по электронной почте писем, приуроченных к завершению проекта. Получив такое письмо, все смогут одновременно и в равной степени почувствовать, что проект закончен и команда выполнила поставленную перед ней задачу. Это письмо также может послужить катализатором празднования выхода нового выпуска.

Празднование нового выпуска

Один из самых замечательных моментов, наступающих после выхода программного продукта, — праздник по случаю нового выпуска. Чувство выполненного долга, после которого приходит праздник, — это что-то потрясающее. Чем дальше отстоят эти моменты во времени, тем меньшее впечатление от праздника. После выпуска ПО можно:

- открыть бутылку шампанского;
- подать всем мороженое прямо на рабочие места;
- пригласить команду на ужин в ресторан;
- устроить совместный поход в кино;
- собраться на пикник в доме одного из участников команды;
- отправиться в боулинг и т.п.

Какое бы мероприятие вы ни выбрали, главное, чтобы все смогли принять о нём участие: некоторые могут и не изъявить желания участвовать в турнире по пейнтболу или в катании на горных велосипедах.

Общественное признание

Общественное признание может стать мощным средством выражения благодарности группам и отдельным участникам за исключительные достижения. Общественное признание может быть как на уровне подразделения, так и на более

высоком — группы или целой компании. Однако независимо от размера команды следует придерживаться некоторых основных правил.

- Должны быть отмечены лишь значительные достижения

Награды заслуживают только усилия, выходящие за рамки должностных обязанностей.

- Необходимо отмечать отличную работу на любом поприще

Следует поощрять отличившихся работников всех отделов, а не только какого-либо одного.

- Излагайте суть достижения коллективу

Не следует предполагать, что все в курсе всех событий. Расскажите немного людям о возникших проблемах и о том, как действия группы или отдельного специалиста помогли справиться с ними.

- Благодарностъ должна быть материальной

Памятный знак или премия сделают поощрение по-настоящему запоминающимся.

Из собственного опыта

Практически каждый месяц в NuMega проводятся общие собрания компании. Сотрудники в полном составе заслушивают новости, поступающие из разных групп. В завершение собрания мы присуждаем награду «Спасителю компании». Ею отмечается беспримерный вклад работника, который помог решить критическую проблему, выйти из затруднительной ситуации или существенно увеличить прибыль. Эта церемония всегда сопровождается рассказом о действиях работника или группы, чтобы убедить всех в том, что награждаемый сыграл важную роль и награда получена заслуженно. Отличившиеся получают на память бейсболки с надписью «Я спас компанию», которые они с гордостью носят или держат в своём кабинете.

Личная благодарность

Личная благодарность — очень эффективный способ продемонстрировать ценность вклада отдельного участника команды в реализацию проекта. Фактически искренняя личная благодарность часто значит для людей даже больше, чем любая форма общественного признания.

Личную благодарность обычно выражают на собраниях, где менеджер проекта или ведущий специалист открыто и искренне благодарит человека за сделанный вклад. Простая фраза вроде «я очень, рад, что в конце работы над проектом вы смогли протестировать программу на других платформах; если бы мы не обнаружили эту ошибку, продукт пришлось бы отозвать» может быть очень важной для того, кто, вложив дополнительные усилия, смог решить ключевую проблему. Это даёт работнику понять, что администрация в курсе его достижений и признательна ему.

К способам выражения личной признательности также можно отнести благодарственное письмо, присылаемое по электронной почте. Надеюсь, вы получали раньше неожиданные благодарственные послания, и ощущения, сопровождающие получение такого письма, вам знакомы.

Премии, подарки и акции компании

Ещё один способ выразить благодарность — наградить отличившегося. Ничто не может так подчеркнуть и подкрепить устную благодарность, как получение премии или подарка. Вручение подарка или премии говорит о том, что хорошая работа замечена, и, что ещё важнее, не осталась без награды. Часто, когда люди знают, что администрация видит, ценит и поощряет отличную работу, они стремятся работать ещё лучше.

Хотя денежные и материальные формы поощрения самые желательные, они не всегда доступны или возможны. Постарайтесь тогда придумать иные способы наградить отличившихся. Почётная грамота или прибавка к отпуску тоже могут быть выражением искренней благодарности.

Памятные фотографии и «пасхальные яйца»

Как было сказано в начале книги, реализованный проект является результатом усилий всей команды. Что может подчеркнуть это лучше, чем групповая фотография после выхода нового выпуска? Подарите такую фотографию каждому участнику команды, а ещё одну повесьте в рамке на видном месте, лучше всего на стене, посвящённой достижениям компании. Эти фотографии свидетельствуют, что в компании всегда видели и ценили командную работу. Пройдёт время, и будет здорово вспомнить, с кем вы работали над разными выпусками, взглянув на эти фотографии.

Кроме того, во многих коллективах любят помещать в программы так называемое «пасхальные яйца» — это скрытые окна, которые можно вызвать определённой комбинацией нажатий клавиш, команд меню и щелчков, содержащие список всех создателей программы, а иногда и их фотографию. Они похожи на заключительные титры кинофильма.

Из собственного опыта

Группа специалиста NuMega, работавших над программой BoundsChecker, поместила в программу «пасхальное яйцо». Те, у кого есть эта программа, могут увидеть его. Для этого нужно вызвать диалоговое окно «О программе» командой меню Help/About, навести указатель на клетчатый значок продукта, затем при нажатой клавише Shift трижды щёлкнуть правой кнопкой.

Что дальше?

Когда все благодарности розданы, пора переключаться на подготовку к следующему проекту. Хотя этот период очень важен, часто о нём забывают. Это самое подходящее время для извлечения уроков из прошлых ошибок и подготовки к решению грядущих задач.

Учимся на ошибках прошлого

Чтобы встретить будущее во всеоружии, следует разобраться в ошибках прошлого. Что удалось? Что нет? Чем больше всего будет отличаться работа над следующим проектом? Определите, в каких продуктах, процессах, технологиях или оборудовании ощущалась острая нехватка на протяжении последних месяцев и проследите, чтобы все это теперь было в наличии.

Классический способ анализа законченного проекта — это обсуждение его на итоговых собраниях. Как правило, такое собрание проводится с участием всей команды вскоре после выхода нового выпуска. Оно служит для обмена мнениями о том, что удалось, а что нет, а также для «мозгового штурма» проблем. Цель такого собраний не в том, чтобы кого-то обвинить или отыскать личные просчёты, а в том, чтобы сообща извлечь уроки из прошлых ошибок и наметить, что нужно сделать во время следующего проекта. Эти собрания идеально подходят для подготовки почвы для грядущих изменений.

Усиление инфраструктуры

Первые дни и недели после выпуска ПО также очень удобны для расширения инфраструктуры — организации новых рабочих процедур, повышения автоматизации, пополнения оборудования и инструментария. Как известно, вносить существенные изменения в эти сферы во время работы над проектом очень трудно. Рассмотрим их поочерёдно.

•

Рабочие процедуры

Как следует изучите все рабочие процедуры: создание ежедневных сборок ПО, базисные тесты, формулирование требований, планирование, оценку практичности и набор антикризисных мер. Как налажен обмен информацией с командой? Адекватны ли планы и методики задачам проекта? Нуждаются ли эти сферы в улучшении?

•

Автоматизация

Часто команды находят степень автоматизации разработки и испытаний программы недостаточной для решения поставленных перед ними задач. Время до начала следующего проекта идеально подходит для повышения автоматизации и навёрстывания упущенного в этой области.

•

Оборудование

Следует использовать полученную возможность для приобретения оборудования, которое потребуется для работы над следующими проектами.

•

Инструментарий

Замена инструментов для управления исходным кодом и отслеживания ошибок во время работы над выпуском, как правило, является ошибкой и всегда требует больше времени (см. главу 5). Но когда проект завершён, можно уделить часть времени оценке новых версий полезных программ и обновлению имеющегося инструментария.

Работа с кадрами

Вложения в кадры не менее важны, чем в инфраструктуру. Как это сделать конкретно?

•

Анализ эффективности работы

По окончании проекта следует проанализировать эффективность работы его участников, Хотя в большинстве компаний такое мероприятие проводится лишь в день приёма сотрудника на работу, индивидуальный анализ работы каждого члена команды по окончании каждого проекта позволяет держать в памяти свежие данные о его эффективности и устранить ряд проблем прежде, чем начнётся работа над следующим проектом.

•

Взаимное обучение

Следует подумать об обмене обязанностями между участниками команды. Очень важно, чтобы при работе над разными фрагментами ПО они обучали друг друга. Плохо, когда разработка какого-либо фрагмента программы полностью зависит от единственного специалиста. Взаимное обучение придаёт большую гибкость планам и позволяет каждому члену команды получить более чёткое представление о многочисленных аспектах проекта.

- Повышение квалификации

Это прекрасное время для повышения квалификации сотрудников. Когда работа над проектом позади, участники команды смогут сосредоточиться на изучении новых технологий или новшеств в уже известных им методиках, появившихся во время работы над последним проектом.

- Отпуска

В любом случае у каждого участника команды должен быть отпуск. Промежуток между проектами лучше всего посвятить семье и личным интересам. Люди стремятся трудиться интенсивнее и много работают сверхурочно, если знают, что смогут хорошенько отдохнуть после завершения проекта.

Те, кто особенно интенсивно работал в течение долгого времени, заслуживают дополнительных выходных. Следует беречь силы тех, кто вносит ключевой вклад во внутренний цикл реализации проекта и давать им дополнительное время для отдыха. Ваша задача — не допуская чрезмерного расслабления, помочь людям восстановить свои силы, чтобы спустя некоторое время они вновь смогли работать с максимальной самоотдачей.

В первоначальный период работы любой компании, когда особенно часто приходится работать сверхурочно, потребность в увеличении времени отдыха после интенсивной работы ощущается особенно остро. В NuMega мы смогли взять месяц оплачиваемого отпуска лишь после пяти лет работы. Все были рады наконец получить компенсацию за все наши сверхурочные. К счастью, все больше компаний признают необходимость увеличенного периода отдыха и осознают то благо, которое он может со временем принести как работнику, так и компании.

Общие проблемы и решения

Далее обсуждается ряд типичных проблем и вопросов, возникающих при использовании описываемых здесь методик, а также их решения.

Чувство опустошённости

Когда проект завершён, у некоторых появляется ощущение опустошённости и разочарования. Возникает чувство, что их вклад и усилия — всё впустую. Поэтому они вновь и вновь берутся за работу, но без особых шансов решить реальные проблемы в масштабах всего проекта или повысить свой личный уровень. Чтобы избежать этой проблемы, следует правильно проводить закрытие проекта, отмечать людей, внёсших основной вклад, проводить итоговые собрания и анализировать эффективность работы участников, которые должны меняться обязанностями и получать достаточно времени для отдыха.

Истощение сил

Истощение сил — самая серьёзная причина возникновения ощущения опустошённости. У «перегоревших» на работе нет ни сил, ни интереса для участия в проекте или даже для выполнения своих профессиональных обязанностей. Истощение развивается со временем и становится настоящей бедой после того, как оно нанесёт свой удар. Основная цель этой главы — обсудить действия, необходимые для предотвращения истощения сил, а не для борьбы с ним после того, как оно проявилось.

Нужно довести проект до конца

Вероятно, многие изложенные в этой главе идеи не новы, но уж слишком часто люди относятся к ним очень легкомысленно. Следует составить конкретный план, чтобы все действия по закрытию были обязательно проведены в полном объёме. Не следует приступать к работе над следующим проектом, не завершив текущий.

Встреча со студентами СГАУ

HAULMONT

Менеджмент в разработке программного обеспечения

Докладчики:

Зоткин Александр (Директор)
Ласкин Иван (Руководитель Направления)

Компания HAULMONT

Центр разработки (свыше 60 чел) – Россия

Представительства: Россия, Великобритания

Собственная платформа разработки и собственные технологии

Современный менеджмент

Признание проектов в Великобритании:

- **National Business Awards 2009**
- **Business Travel Awards 2010**

Направления деятельности

- Разработка, внедрение и сопровождение корпоративных информационных систем (КИС):
  - автоматизация ключевых процессов предприятия
  - системы автоматического принятия решений и оптимизации ресурсов
  - Обеспечение непрерывности бизнеса через построение высоко-отказоустойчивых решений.
- Автоматизация документооборота (СЭД):
  - система управления документами и задачами «ТЕЗИС»
  - автоматизация уникальных бизнес-процессов
- Разработка интерактивных порталов

Основная цель менеджера проекта

Сдат ~~чеством~~

~~т?~~

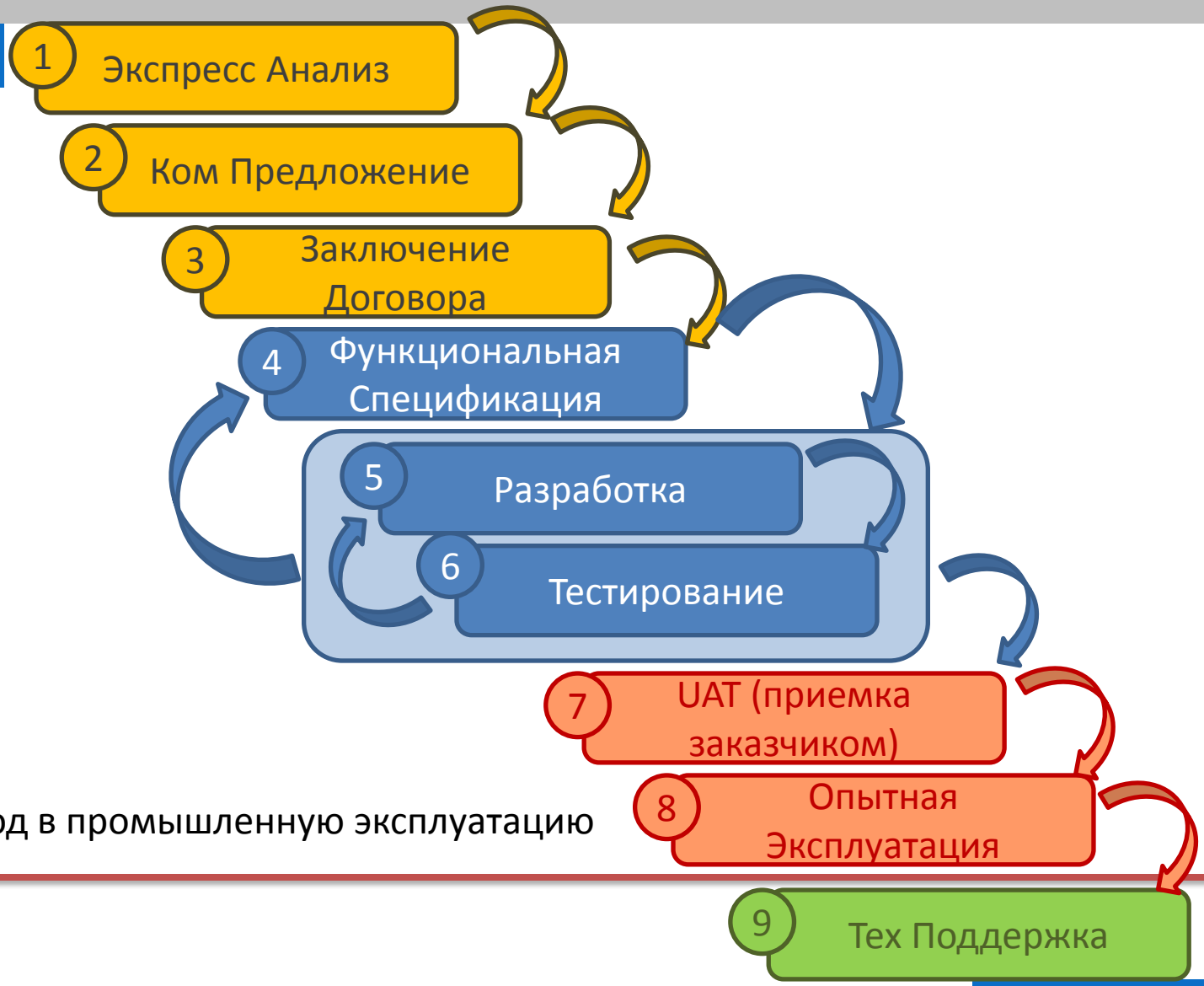
Довольный заказчик

Из чего состоит процесс управления проектами

- Работа с заказником
  - Установление эффективных коммуникаций с заказчиком
  - Управление ожиданиями заказчика
  - Управление рамками проекта
- Управление командой
  - Планирование
  - Обеспечение необходимыми ресурсами
  - Мотивация
- Управление рисками
- Управление бюджетом проекта
- Управление сроками

Управление проектами под заказ

Основные Этапы Проекта



Ввод в промышленную эксплуатацию

Вопросы

Управление проектами внедрения продукта

Основные сценарии внедрения продукта

- Внедрение «коробочного» решения

- Разработка проекта на базе системы ТЕЗИС

Внедрение типового решения

- Подписание договора
- Установка Системы на сервер
- Обучение администратора
- Обучение пользователей
- Опытная эксплуатация
- Передача Системы в промышленную эксплуатацию

Риски типового внедрения

Риск	Способы снижения
• Саботаж проекта со стороны пользователей	• Приказ о начале внедрения
• «Все внедрили, а никто не работает»	• Выявление «союзников», новаторов в компании
	• Подключение менеджмента на ранних стадиях внедрения
• Не соответствие системы требованиям Заказчика	• Вовлечение специалиста по внедрению на этапе Presale

Примеры проектов на базе Системы «ТЕЗИС»

НК «Альянс» - разработка и внедрение СЭД:

- вертикально-интегрированная нефтяная компания, работающая в России и Казахстане.
- обладает 638 миллионов баррелей доказанных и вероятных запасов нефти
- контролирует Хабаровский НПЗ и сеть нефтебаз и АЗС на Дальнем Востоке России.

ООО «КМ-Эксперт» (генподрядчик ЗАО «Группа ЧТПЗ»)

- автоматизация процесса согласования проектно-сметной документации, КИД

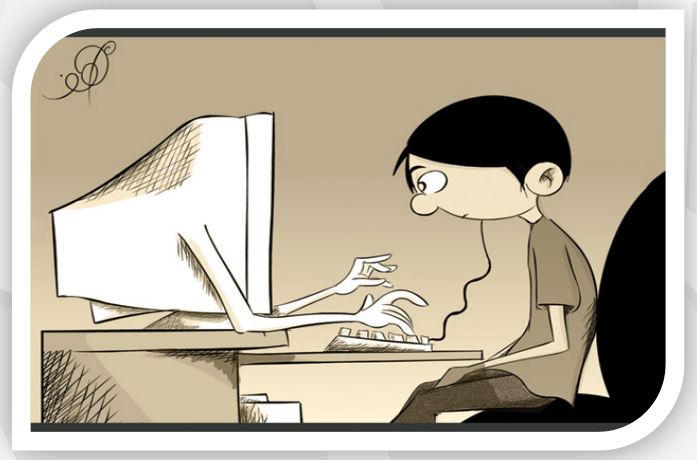
- Проект по строительству объектов цеха «Высота 239»

Разработка проекта 1

- Заключение договора на предпроектное обследование
- Предпроектное обследование
- Согласование функциональной спецификации
- Разработка
- Тестирование

Разработка проекта 2

- Опытная эксплуатация
- Обучение администратора системы
- Обучение пользователей системы
- Приказ о начале промышленной эксплуатации
- Техническая поддержка



Снижение рисков проекта

- Выделенный менеджер проекта со стороны Заказчика
- Согласованный календарный план работ
- Еженедельный отчет о текущем состоянии проекта для Заказчика
- «Непрерывная» работа с Командой

Вопросы

DMITRI Y. MANIN

3127 Bryant St., Palo Alto, CA 94306 *650-575-1506; manin@pobox.com*

Research summary

My research publications are dated before 1998, when I worked in academic institutions, and after 2005, when I resumed publishing as an independent researcher in the field of quantitative linguistics. My work in industry since 1997 was in engineering and produced two software-related patents.

The large paper [3] and its spin-off [2] are devoted to Zipf's law, one of the most enigmatic and controversial regularities in linguistics. It states that if the words of a language are ranked in the order of decreasing frequency in texts, the frequency of a word is inversely proportional to its rank. Zipf's law itself and related properties of texts (such as the number of hapaxes in a corpus) are directly relevant for such enterprises as Web search.

Zipf's law is considered by some as "linguistically shallow" (B. Mandelbrot), while others perceive it as carrying an important information about the language. None of the existing models for Zipf's law in linguistics, however, were able to reveal this information. I argue in [3] (following French linguist P. Guiraud) that Zipf's law acts on the semantic level, and affects word frequencies only indirectly, through semantics. I presume that word frequency is proportional to the extent of its meaning, so that words that have less specialized, more generic meanings, are usable in more speech situations, and so more frequently. I then review the mechanisms and directions of meaning change and conclude that 1) meanings have a tendency to expand, and 2) meanings of similar extent compete. This last idea is known in linguistics as avoidance of excessive synonymy: the language tends to semantically differentiate words that come too close to meaning the same thing.

Given this dynamics, I argue that it will produce the Zipfian distribution. I also construct two simplified numerical models which simulate the evolution of meanings and indeed robustly generate Zipf's distribution. Finally, I analyze word frequency in Russian to substantiate the relationship between meaning extent and frequency. This is done by showing that the frequency of a word tends to be equal to the net frequency of all its hyponyms (more specific words, such as $strawberry$ vs. $berry$). In the process, I propose a novel tool for elucidating word semantics using Web search. The structure of the natural language's lexicon that is elucidated in this work can be relevant to many problems in NLP.

The paper [2] is devoted to the analysis of the widely known model due to Mandelbrot, where Zipf's law is obtained by optimizing the cost of information transmission. I demonstrate that the two parameters of this model are not independent, as is usually believed, and as a result it can not account for the actual word frequency distributions.

In preprint [4], I analyze long-range correlations of letters in natural texts. The power-law statistics that they exhibit are believed by many to reflect "complexity" of the underlying object (i.e. the text). I demonstrated however that these statistical properties are preserved when letters of the text are randomly shuffled in a moving window, which completely destroys the text with all its associated complexity. I further study the origin of the correlations and show that they are due to the small subset of "topical words" that slowly changes along the text.

In the 2006 article [5], I present one particular result of a large-scale experiment in quantitative linguistics and poetics that I am conducting since 2003. I designed and implemented this experiment as a Web-based literary game in Russian to obtain quantitative data on literary text perception. It involves subjects guessing words in fragments of poetry and prose. Over 1 million data points have been collected so far. The result presented in [5] is a strikingly linear dependence

of the log probability to guess an omitted word on the length of the word. It holds equally for poetry and prose. It is interpreted as an evidence that language tends to even out the information rate. This result can be relevant for Web search and text analysis, since it helps to estimate the information weight of a word in text.

The most recent paper [1] analyzes data gathered in the above experiment to answer a long-standing question of whether free verse (verse without rhymes and meter) is any different from prose arbitrarily split into short lines. Until now, scholars could rely only on their intuition and insight when discussing such matters. Hard experimental data allows to provide some definitive answers and generate new insights. It turns out, in particular, that free verse is indeed different from both prose and metered verse, being like the former in some respects and like the latter in others.

In article [6], the concept of entropy was applied to quantify the mixing quality of a fuel jet injected into a combustion chamber of a turbine.

Article [8] demonstrates the fruitfulness of a cross-disciplinary approach. In it, I proposed a novel method for analyzing and simulating large populations of neurons in the mammal primary visual cortex. The method uses the concept of *kinetic equation*, well-known in statistical physics, but never before applied in neurobiology. This work spawned a significant amount of subsequent research in the field.

In the unpublished writeup [7] I considered the distribution of Web pages by their hit rate (not to be confused with the distribution by the number of links) within a large and inhomogeneous Russian entertainment site. The distribution turned out to satisfy a clean power law. To explain this fact, I hypothesized that it reflects the structure of the graph formed by pages linked together. I introduced the notion of *graph dimension*, based on the number of pages reachable from the home page in a finite number of clicks ("the volume of a sphere of a given radius"). This allowed to estimate the dimension of the actual graph, which turned out to be about 7.

Most of my preceding work conducted at Moscow Institute of Atmospheric Physics was focused on two related topics: coherent vortical structures in the Earth's atmosphere and the behavior of a rotating fluid in cylindrical and conical reservoirs. The atmosphere, when looked at from the global perspective, is a thin layer of rotating fluid, and rotating fluid is a very peculiar object that can be likened to a spinning top with an infinite number of dimensions. It has a very rich dynamics and is prone to form stable vortical structures, i.e. cyclones and anticyclones directly affecting our weather. As a theoretician, I worked in close contact with experimentalists and was involved both in building mathematical models and interpreting experimental data. In one of my favorite works from that period [24] I estimated the size of large-scale atmospheric vortices and re-interpreted experimental data from a number of previously published papers to demonstrate that all of them support my theoretical result.

In summary, my publication history demonstrates my experience in many different fields, the ability to work independently and apply cross-domain intuitions to achieve novel results, as well as the result-oriented approach.

References

1. *Manin, D.Yu.* 2009. Chopped-up Prose or Liberated Verse? An experimental study of Russian *vers libre*. Accepted for publication in *Modern Philology*.

2. *Manin, D.Yu.* 2009. Can Mandelbrot model explain Zipf's law? *Quantitative Linguistics*, **16** (3) 274–285.

3. *Manin, D.Yu.* 2008. Zipf's law and avoidance of excessive synonymy. *Cognitive Science Journal*, **32** (7) 1075–1098. Preprint available at http://arxiv.org/abs/0710.0105.

4. *Manin, D.Yu.* 2008. On the nature of long-range letter correlations in texts. arXiv:0809.0103

5. *Manin, D.Yu.* 2006. Experiments on predictability of word in context and information rate in natural language. J. Information Processes (electronic publication, http://www.jip.ru/2006/229-236-2006.pdf), **6** (3), 229-236.

6. *Everson, R., Manin, D., Winter, M., and Sirovich, L.* 1998. Quantification of Mixing and Mixing Rate from Experimental Observations. *AIAA Journal*, **36** (2), 121

7. *Manin, D.* 1998. What is the dimension of the Platonic world of ideas? Unpublished, http://centrolit.kulichki.com/centrolit/manin/platonic_dim.html.

8. *Knight, B.W., Manin, D., and Sirovich, L.* 1996. Dynamical models of interacting neuron populations. In: *Symposium on Robotics and Cybernetics; Computational Engineering in Systems Applications.* (Gerf, E.C. ed) Cite Scientifique, Lille, France.

9. *Sirovich, L., Everson, R., and Manin, D.* 1995 Turbulent Spectrum of the Earth's Ozone Field *Phys. Rev. Lett.*, **74** (13), 2611–2614.

10. *Manin, D.Yu. and Nazarenko S.V.* Nonlinear interaction of small-scale Rossby waves with an intense large-scale zonal flow. *Phys. Fluids*, **1** (3), 1158–1167.

11. *Dolzhanskiy, F.V., Manin, D.Yu.*, 1994. Effect Of The Turbulent Ekman Layer On The Dynamics Of Large-Scale Atmospheric Motions. *Transactions (doklady) of the USSR Academy of Sciences.* **323A** (3), 22

12. *Danilov, S.D., Dolzhanskii, F.V., Manin, D.Yu.*, 1993. Dynamics of large-scale flows with turbulent Ekman layer and their stability. *Annales geophysicae. Atmospheres, hydrospheres.* **11** (2/3), 104

13. *Dolzhanskii F.V., Krymov V.A., Manin D.Yu.*, 1993. Computer-aided analysis of experimental flow fields. *Russian J. Comp. Mech.*, **1** (1), 95-106.

14. *Krymov, V.A., Manin, D.Yu.*, 1992. Reconstruction of the External Force Field from the Velocity Field of a Quasi-Two-Dimensional Flow. *Izvestiia. Atmospheric and oceanic physics.*, **28** (2), 95.

15. *Dolzhanskii, F.V., Krymov, V.A. and Manin, D.Yu.*, 1992. An advanced experimental investigation of quasi two-dimensional shear flows. *J. Fluid Mech.* **241**, 705–722.

16. *Manin, D.Yu.*, 1992. A study of repeated vortex mergers in a forced quasi 2D shear flow. *Phys. Fluids A*, **4** (8), 1715–1723.

17. *Dolzhanskiy, F.V., Manin, D.*, 1992. Turbulent Ekman Layer and the Equation of Transformation of the Potential Vorticity. *Izvestiia. Atmospheric and oceanic physics.* **28** (1), 1.

18. *Dolzhanskij, F.V. and Manin, D.Yu.*, 1992. The effect of turbulent Ekman layer on large-scale atmospheric dynamics. *Sov. Phys. Doklady*, **223** (6).

19. *Dolzhanskii, F.V., Krymov, V.A. and Manin, D.Yu.*, 1992. Nonlinear spin-up and spin-down in a cylinder with small ratio of height to depth. *J. Fluid Mech.*, **234**, 473–486.

20. *Krymov, V. A., Manin, D. Yu.*, 1991. Experimental Study of Velocity Fields for Quasi-Two-Dimensional Flows. *Izvestiya. Atmospheric and oceanic physics.* **27** (7), 538

21. *Dolzhanskii, F. V. and Manin, D. Yu.*, 1991. *On the effect of turbulent Ekman layer on global atmospheric dynamics*. Preprint No.6, Inst. Atmos. Phys., Moscow, 10 p.

22. *Dolzhanskii, F. V., Krymov, V. A. and Manin, D. Yu.*, 1991. Quasi two-dimensional coherent structures. In *Nonlinear dynamics of structures*. World Scientific, Singapore, p. 1–20.

23. *Manin, D. Yu. and Chernous'ko, Yu. L.*, 1990. Experimental investigation of the stability of a quasi two-dimensional jet flow produced in a rotating fluid by the method of sources and sinks. *Izvestiya. Atmospheric and oceanic physics.*, **26** (5), 327.

24. *Manin, D. Yu.*, 1990. Characteristic Size of Vortices in Developed Quasi-Two-Dimensional Flows. *Izvestiia. Atmospheric and oceanic physics.*, **26** (6), 426.

25. *Dolzhanskij, F. V. and Manin, D. Yu.*, 1990. Laboratory simulations of atmospheric circulation and free shear layers. *Izv. AN SSSR. Fiz. Atmos. Okeana*, **26** (12), 1282–1288.

26. *Dolzhanskii, F. V., Krymov, V. A. and Manin, D. Yu.*, 1990. Stability and coherent structures in quasi-two-dimensional shear flows. *Sov. Phys. Uspekhi*, **33** (7), 495–520.

27. *Manin, D. Yu.*, 1990. Stability of quasi two-dimensional shear flows in the presence of the beta effect and external friction. *Izvestiya, Atmos. Ocean. Phys.*, **25** (8), 593–598.

28. *Manin, D. Yu.*, 1989. *On the characteristic length scale of vortices in developed quasi two-dimensional flows*. Preprint No.6, Inst. Atmos. Phys., Moscow.

29. *Krymov, V. A. and Manin, D. Yu.*, 1989. Linear and nonlinear stability of quasi-two-dimensional jet flows with external friction. *Izvestiya, Atmos. Ocean. Phys.*, **25** (3), 172–178.

30. *Manin, D. Yu.*, 1989. Stability and supercritical regimes of quasi two-dimensional shear flow in the presence of external friction (theory) *Fluid dynamics*, **24** (2), 177.

31. *Krymov, V. A. and Manin, D. Yu.*, 1986. Spin-down of a viscous fluid between infinite cones. *Izv. AN SSSR. Mekh. zhidk. gaza*, (4), 37–44.

32. *Krymov, V. A. and Manin, D. Yu.*, 1986. Spin-down of a viscous fluid in a low cylinder at high Reynolds numbers. *Izv. AN SSSR. Mekh. zhidk. gaza*, (3), 39–46.

33. *Manin, D. Yu. and Petviashvili, V. I.*, 1983. Self-focusing of a magnetosonic wave across the magnetic field. *JETP Lett.*, **38** (9), 517–520.