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)

Методы проектирования и поддержки требований к программному обеспечению

Электронный учебно-методический комплекс
по дисциплине в LMS Moodle Электронный учебно-методический комплекс
по дисциплине в LMSMoodle

Работа выполнена по мероприятию блока 1 «Совершенствование
образовательной деятельности» Программы развития СГАУ
на 2009 – 2018 годы по проекту «Разработка магистерской программы
«Программное обеспечение мобильных устройств» по направлению
230100.68 Информатика и вычислительная техника»
Соглашение № 1/12 от 3.06.2013 г.

САМАРА
2013

УДК004.9(075)
М545

Автор-составитель: **Иващенко Антон Владимирович**

Методы проектирования и поддержки требований к программному обеспечению [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle/ Минобрнауки России, Самар.гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост.. А.В. Иващенко. - Электрон. текстовые и граф. дан.- Самара, 2013. –1 эл. опт.диск (CD-ROM).

В состав учебно-методического комплекса входят:
1. Курс лекций семестр А.
2. Вопросы к экзамену семестр А.
3. Рабочая программа курса.

УМКД «Методы проектирования и поддержки требований к программному обеспечению» предназначен для студентов факультета информатики, обучающихся по направлению подготовки магистров 230100.68 «Информатика и вычислительная техника» в семестре А.

УМКД разработан на кафедре информационных систем и технологий.

Лекция 1. Общие понятия менеджмента разработки ПО

1.1 Общие понятия менеджмента разработки ПО

Разработка программного обеспечения в большинстве случаев должна рассматриваться как коллективный труд специалистов, направленный на удовлетворение потребности пользователей в автоматизации их деятельности. Это процесс, порою длительный, связывающий производственными и иными отношениями тех, кого в той или иной степени можно рассматривать в качестве производителей программы. Как и любой труд, тесно связанный с неоднозначными потребностями тех, кто будет использовать продукты труда, необходимым элементом разработки программ является решение задач изучения пользователей, с одной стороны, а с другой – обеспечения обратной связи с ними, направляющей производство. Это составляющие, из которых формируются главные задачи управления производством программ. Чаще всего решение таких задач осуществляется руководителем, или, как принято говорить, менеджером проекта.

Понятие «менеджер проекта» необязательно соотносится с конкретной персоной, отвечающей за управление производством программной системы в целом. В небольшом проекте такое единоначалие чаще всего оправданно: оно позволяет концентрировать все нити управления, исключает проблемы внутреннего для проекта согласования противоречий, обеспечивает централизованную ответственность за проект перед теми, кто заинтересован в его выполнении. Однако по мере перехода к более масштабным проектам менеджерские обязанности становится невозможно концентрировать в одних руках. Обычно в таких случаях поступают в соответствии с одной из двух схем организации производства.

- Первая схема – это образование службы менеджера, состоящей из его помощников, которым он может поручать различные задания, освобождая себя от рутины постоянного контроля.
- Для еще более сложных проектов появляется необходимость следовать второй схеме, т.е. образовывать группу менеджеров, ответственных за разные разграниченные сферы проекта.

В этой схеме централизация достигается путем назначения главного менеджера проекта, который делегирует полномочия менеджерам по направлениям.

Рисунок 1 иллюстрирует три схемы организации менеджмента проекта. Здесь стрелки обозначают связи участников реализации проекта, обусловленные их взаимодействием с менеджером, жирность контура значков отражает менеджерские обязанности сотрудников. Как видно из рисунка, в схеме со службой менеджера помощники по своему статусу являются обычными работниками, тогда как при делегировании полномочий менеджеры по направлениям получают соответствующие полномочия в своих сферах ответственности (обозначены пунктирными овалами).

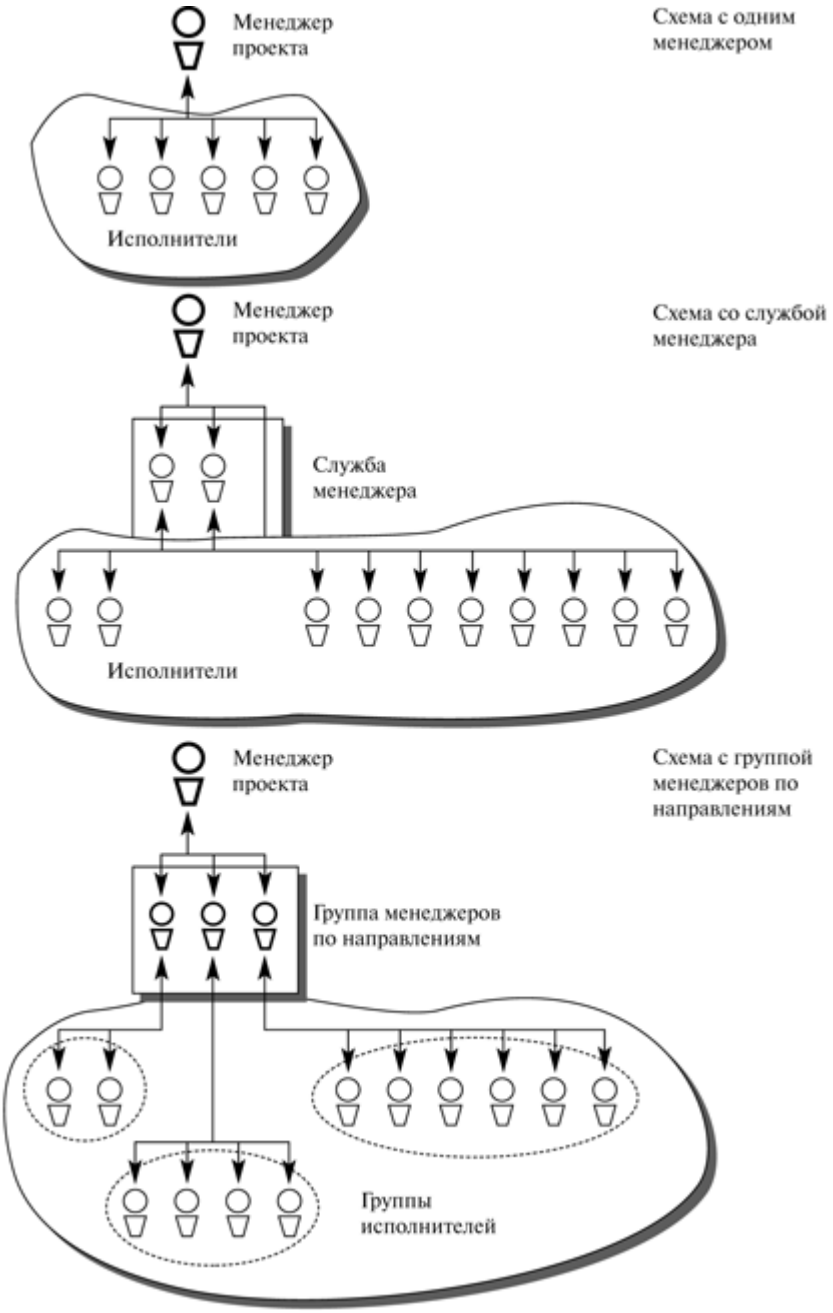


Рисунок 1. Схемы организации менеджмента проекта

Делегирование можно считать основным инструментом разделения труда в проекте, когда есть ответственность за некоторую функцию (работу и др.), но для ее выполнения нет собственных ресурсов, а потому приходится прибегать к помощи. Собственно говоря, различие схем работы менеджера связано с использованием механизмов поручений или делегирования.

В коллективах, выполняющих программные проекты, возможны самые разнообразные организационные структуры. Так, для сложных, объемных по трудозатратам проектов иерархические цепочки «менеджер – менеджер по направлению – исполнитель работ» целесообразно увеличивать, дифференцируя работы и сферы ответственности для некоторых менеджеров по направлению.

Иногда оправданно делегирование всех менеджерских обязанностей и полномочий менеджерам по направлениям. В результате ответственность за проект несет не один человек, она распределяется по менеджерской группе, т.е. возникает коллективная,

деперсонифицированная ответственность. Возможны схемы, когда вместо менеджеров по направлениям деперсонифицированная ответственность назначается группе в целом, т.е. менеджер проекта выдает задания, контролирует их выполнение, осуществляет другие функции, обращаясь ко всей группе, внутри которой уже распределяются обязанности, в том числе и обязанности менеджера по направлению.

Для небольших групп возможна следующая организация работ: обязанности главного менеджера распределяются по команде разработчиков, которая за счет внутренних механизмов решает, как планировать работы, как их распределять и контролировать. Это характерно для так называемого подхода быстрой разработки (agile development) программных систем, объединившего в себе несколько методологий программирования, которые отказываются от многих принципов традиционного менеджмента.

По ряду причин может оказаться целесообразным сочетание схем организации менеджмента проекта. В результате появляются, например, коллективы, в которых главный менеджер берет на себя функции менеджера определенного направления, тем самым он отвечает и за весь проект в целом, и за некоторую его часть.

В работе менеджера всегда присутствуют и неразрывно связаны друг с другом два аспекта:

- управление проектом как деятельность, продвигающая процесс производства к определенным целям,
- руководство людьми, участниками разработки.

Для первого из этих аспектов можно указать методические приемы и иногда даже организационные технологии, обеспечивающие получение приемлемых результатов в заданное и вполне оцениваемое время. Но что касается руководства людьми, то оно всегда было и остается в значительной степени искусством. Тем не менее, важно подчеркнуть, что управление проектом ведется через персоналии его участников, через работу с ними и их взаимодействие между собой, а потому игнорировать аспект руководства для любого проекта ни в коем случае нельзя. Ставя задачу изучения менеджмента, приходится обсуждать как управление, так и руководство. Эта двойственность характерна для любого менеджмента, но для менеджмента программных проектов она играет решающую роль, поскольку данная отрасль направлена на получение не материальных, а идеальных «мыслительных» продуктов, так называемых артефактов.

С организационной точки зрения в разработке программного обеспечения можно выделить три варианта целей, определяющие деятельность менеджера:

- Во-первых, это производство программ не для продажи, напрямую не связанное с получением дохода.
- Во-вторых, это производство рыночного продукта, обеспечивающего прибыль за счет распространения (продажи) получаемых результатов.
- В-третьих, разработка ведется под заказ, когда все производство программы, от стадии замысла до передачи в эксплуатацию, финансируется внешними по отношению к разработчикам, но весьма заинтересованными агентами, обычно называемыми заказчиками.

Варианты существенно различаются и по уровню ответственности разработки, и по требованиям к качеству продукции, и по другим параметрам, которые необходимо отслеживать. Однако, если отвлечься от персоналий и абстрагироваться от различий ресурсов, необходимых для реализации программ в каждом из этих случаев, то в менеджменте производства программ можно найти много общего, того, что в любом случае должен или не должен делать руководитель проекта.

Главная и постоянная задача менеджмента разработки программного обеспечения – продвижение проекта к обозначенным в начале его развития результатам. Если оставить в стороне приемы и методы, с помощью которых достигается решение этой задачи, то она сводится к распределению и контролю эффективного использования имеющихся ресурсов проекта: времени, финансов, технических средств и производственного потенциала работников. Множественность критериев, необходимость согласования интересов участников проекта и его заказчиков, разнообразие видов деятельности, составляющих развитие проекта, – вот тот организационный и производственный контекст, который вынужден учитывать менеджер.

На фоне получения программного продукта как результата появляется ряд дополнительных задач, за решение которых должен отвечать менеджер проекта. Здесь уместно отметить два дополнительных направления развития.

Характерной особенностью разработки программного обеспечения является стремление к переиспользованию ранее созданных программных компонентов. Задачи переиспользования – это:

- во-первых, определение программных продуктов или каких-либо иных изделий и инструментов, имеющихся в распоряжении разработчиков, использование которых могло бы способствовать прогрессу развития проекта,
- а во-вторых, выявление компонентов данного проекта, которые было бы полезно задействовать в других разработках.

Второе дополнительное направление – это задача распространения построенного программного продукта. Если с самого начала не рассматривать ее и относить распространение лишь к этапам, следующим за разработкой, есть опасность сделать не то, что нужно потребителю продукции, и выпустить из рук рычаги, с помощью которых можно влиять на сферу возможного применения создаваемых программных продуктов.

Получение программного продукта как результата развития проекта – это процесс, который регламентируется и направляется пользовательскими потребностями, возможностями применяемого оборудования и другими условиями, как внешними по отношению к проекту, так и внутренними. Все эти аспекты принято рассматривать как требования к проекту. И. Соммервилл определяет два класса требований:

- **Пользовательские требования** – описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на нее.
- **Системные требования** – детализированное описание системных функций и ограничений, которое иногда называют функциональной спецификацией. Оно служит основой для заключения контракта между покупателем системы и разработчиками программного обеспечения.

Ориентируя свое определение на то, что приходится различать требования разных уровней, Соммервилл вводит еще один класс требований:

- **Проектная системная спецификация** – обобщенное описание структуры программной системы, которое будет основой для более детализированного проектирования системы и ее последующей реализации.

Эта спецификация дополняет и детализирует спецификацию системных требований.

Итак, с точки зрения целей программный проект можно рассматривать как деятельность по реализации системы, удовлетворяющей вполне определенным требованиям. Требования не обязательно формулируются все сразу. Более того, очень часто они поступают при развитии проекта и даже уже при использовании полученной программной системы. И это

характерно практически для всех программных разработок. Естественно, деятельность менеджера должна учитывать, что работа над проектом ведется в условиях большой неопределенности относительно конечного результата.

Перечисленные аспекты разработки программного обеспечения определяют специфику этой деятельности в самых общих чертах с точки зрения конкретного ее участника – менеджера программного проекта. Чтобы разобраться в том, какие задачи решает менеджер для организации целенаправленного эффективного развития проекта, мы рассмотрим два вопроса:

- кто участвует в разработке;
- какие этапы проходит проект в своем развитии.

Первый из них поможет выявить функции менеджера в коллективе разработчиков и место этих функций среди других выполняемых при развитии проекта мероприятий. Он имеет два аспекта, непосредственно связанные с проведенным выше разграничением между управлением и руководством.

С точки зрения управления участники проекта – это абстрактные действующие агенты, которые выполняют заданные функции. Здесь под функцией понимается некое действие, при выполнении которого потребляются определенные ресурсы и производится определенный результат. Функциональный взгляд на участников разработки проекта делает их взаимозаменяемыми, обезличенными в пределах компетенции, соответствующей выполнимости функции. Он приводит к понятию роли, назначаемой работнику для выполнения соответствующих обязанностей.

Говоря о руководстве, нужно рассматривать персоналии, которым назначаются роли в проекте. Пожалуй, самая главная задача руководства – это формирование дееспособного коллектива, который в состоянии выполнить данный проект. Понятно, что основным требованием к коллективу является его компетентность и квалификационная обеспеченность, необходимые для выполнения проектного задания. Но если не отработаны способы общения членов коллектива между собой, с руководством, с заказчиками, то никакая компетентность не поможет. В то же время, как показывает практика, недостаточная стартовая квалификация коллектива может успешно преодолеваться совместным обучением сотрудников, проводимым параллельно с основной работой.

Ключевым качеством коллектива, определяющим его успешность, является слаженность. В идеальном коллективе все понимают друг друга с полуслова, есть взаимопонимание и уважение, не происходят или сведены к минимуму внутренние конфликты и противоречия. В реальности менеджеру редко приходится иметь дело с таким коллективом, а значит, необходимы меры, способствующие не только росту индивидуальной квалификации работников, но и дееспособности формируемой команды в целом. Эти меры следует рассматривать в качестве задач менеджера как руководителя коллектива.

Поскольку развитие проекта – работа, существенно зависящая от внешних по отношению к коллективу факторов, определяя место менеджера в коллективе разработчиков, полезно расширить круг участников этого процесса, включив в него, с одной стороны, заказчиков программного продукта, а с другой – руководство организации (компании, фирмы), под эгидой которой выполняется проект.

Вопрос об этапах развития проекта обозначает задачи менеджера как управляющего проектом. Он ставится, чтобы раскрыть, как распределяются функции менеджера и других исполнителей во времени. Неоднородность работ, выполняемых при производстве программных продуктов, зависимость этих работ друг от друга, коллективный характер их выполнения – вот основания для поэтапной организации развития проекта с выставлением заданий на этапы и менеджерским контролем хода работ. Все это связывается с понятием жизненного цикла программного обеспечения.

К настоящему времени вполне сложилось представление о том, на какие этапы разбивается развитие проекта, какие функции связываются с выполнением этапов, как должен осуществляться контроль. По сути, *жизненный цикл* – это процесс конструирования программного обеспечения. Требования к проекту определяют цели и регламент данного процесса, а разработчики являются основной его движущей силой и одним из главных ресурсов. Другие главные ресурсы проекта – это время и финансы, выделяемые для выполнения проектного задания. Кроме главных ресурсов проекта, можно говорить о его производных и вспомогательных ресурсах. К производным ресурсам относятся технические средства, а также программы, используемые в качестве инструментов, заготовок и включаемых компонентов, а к вспомогательным – тепло и электроэнергия, служебные помещения, технические и организационные средства, которые непосредственно с процессом производства не связаны, но способствуют его успешному выполнению. Разделение неглавных ресурсов на производные и вспомогательные во многом зависит от специфики выполняемого проекта.

Так, телефон, вообще говоря, нужно относить к вспомогательным ресурсам, но если проект связан, например, с автоматизацией работы телефонной станции, то этот ресурс следует считать производным.

Трудовой ресурс хорошего проекта является консервативным, т.е. не очень меняющимся в ходе выполнения проекта. При плохой организации дела есть риск текучести кадров, которая для программных проектов отрицательно сказывается на результатах. Время – ресурс невосполнимый, а потому планирование времени является одной из главных забот менеджера. Финансы – основной расходуемый ресурс, который, к сожалению, всегда ограничен. В частности, они идут не только на обеспечение заработной платы работников, но и расходуются на технические средства и используемые программы.

В менеджменте программных проектов приходится решать задачу составления и соблюдения оптимального баланса расходования ресурсов. Эти задачи характерны для деятельности менеджера практически любого проекта. Разумеется, на нее влияют и специфика проекта, и условия его выполнения. Как следствие, существует множество подходов, методов, методик и даже технологий, поддерживающих менеджмент.

Таким образом, при обсуждении задач менеджера в разработке программных изделий мы сталкиваемся с тремя взаимосвязанными направлениями его деятельности.

- Первое направление – это те функции, которые необходимо выполнять для успешного развития проекта. Здесь следует выяснить, какие роли сотрудников требуются для данного проекта.
- Второе направление – планирование и контроль хода проекта в соответствии с жизненным циклом создаваемого программного обеспечения.
- Наконец, третье направление определяется кругом задач формирования коллектива и, в частности, кадровым обеспечением проекта.

1.2 Проект и проектная деятельность

Проект определяется как уникальный комплекс взаимосвязанных мероприятий, направленных на достижение конкретной цели при определенных требованиях к срокам, бюджету и качеству ожидаемых результатов.

Проект как вид деятельности отличается следующими особенностями:

1) направлен на достижение конкретной цели (целей);
2) включает в себя выполнение взаимосвязанных действий, называемых задачами;
3) имеет ограничение по времени выполнения;

4) использует ограниченные ресурсы: людские, финансовые, материальные;

5) большинство проектов имеет свои уникальные особенности.

В отличие от традиционного управления организацией проектное управление охватывает не весь жизненный цикл предприятия. Проект длится до тех пор, пока не будет достигнута определенная цель (например, создание уникального продукта), и имеет определенные даты начала и окончания. Каждый проект характеризуется своим масштабом, который означает сочетание целей, планируемых затрат ресурсов и времени.

Сложность управления задачами и ресурсами в рамках проекта предполагает наличие адекватной системы управления проектом (Project Management).

Управление проектом – это процесс планирования, организации и контроля над состоянием задач и ресурсов проекта, направленный на своевременное достижение цели проекта.

Проектирование информационных систем включает в себя большое количество взаимосвязанных задач, в их решении могут участвовать большие коллективы разработчиков. Организация процесса проектирования ИС отличается значительной сложностью, которая обусловлена следующими причинами:

1) масштабностью и длительностью разработки ИС;
2) взаимосвязью в рамках системы различных по своей природе объектов (информационные, программные, технические средства, математические модели, методы и средства проектирования и др.);
3) различиями в жизненном цикле элементов системы;
4) индивидуальностью проекта, обусловленной спецификой объекта проектирования;
5) необходимостью коллективного характера работы над проектом специалистов разной специализации и квалификации.

Вследствие этого проектирование ИС предполагает использование проектного управления.

1.3 Составляющие проектного плана

План проекта представляет собой модель, описывающую реальный проект в терминах задач, ресурсов, сроков, затрат.

Задача (task) - деятельность, осуществляемая в рамках проекта, для достижения определенного результата. Задачи являются основными блоками, из которых строится любой проект, они представляют работу, которую нужно выполнить для достижения поставленной цели. Во всем проекте набор задач характеризуется их логической последовательностью, а каждая задача - длительностью и требованиями к ресурсам. **Ресурсы** - исполнители, оборудование и материалы, необходимые для выполнения задачи.

Назначения - связь конкретной задачи с ресурсами, выделенными для ее выполнения.

Проект, как правило, содержит большое количество задач, поэтому весь набор задач необходимо представить в виде укрупненных групп, логически связанных между собой. Так формируются суммарные задачи (фазы)

Суммарная задача (фаза, summary task) - состоит из нескольких задач. Результат фазы обобщает (суммирует) результаты задач, входящих в нее. Суммарная задача может содержать в себе как задачи, так и другие суммарные задачи.

Веха (milestone) - задача, достижение результата которой особенно важно для проекта. Вехой может быть завершающая задача фазы. Как правило, веха используется для обозначения окончания основных этапов проекта.

Трудозатраты (work) - для задач: объем работ (в единицах рабочего времени) необходимый ресурсу (исполнителю) для выполнения задачи.

Длительность задачи (duration) - время, которое запланировано для работы над задачей.

Трудозатраты отличаются от длительности задачи. Ресурсу может потребоваться 24 часа на выполнение задачи, а длительность задачи - 8 часов. Это означает, что на выполнение данной задачи необходимо назначить не менее трех исполнителей. После установления списка задач проекта, длительностей задач, необходимо указать, как задачи взаимосвязаны друг с другом, их логическую зависимость.

Зависимости и связи - определяют логику связи одной задачи с другой, показывая, как одна задача влияет на другую. Например, задача №2 начинается только когда закончится задача №1, или задача №1 и задача №2 начинаются обязательно в одно время.

1.4 Жизненный цикл проекта: этапы разработки ПО

Очевидно, что функции, выполняемые разработчиками проекта, в ходе его развития претерпевают изменения, как, впрочем, и сам проект. Сначала он существует в виде заявки на разработку, затем – как функциональные и технические требования, далее – как спецификации разрабатываемого изделия, набор программных модулей, скомпонованная из модулей система и т.д. Этот перечень можно рассматривать как один из примеров модели жизненного цикла программного изделия, т.е. представления эволюции разработки и последующего использования программной системы.

Жизненный цикл следует рассматривать как основу деятельности менеджера программного проекта: с ним связываются и цели проекта – окончательные и промежуточные, распределение и контроль расходования ресурсов, а также все другие аспекты управления развитием проекта. Прежде всего эта привязка обусловлена разбиением производства любой программы на этапы, которые ассоциируются с определенными видами работ или функций, выполняемых разработчиками в тот или иной момент развития проекта. Этапы характеризуются направленностью выполняемых функций на достижение локальных (для этапа) целей проекта. Необходимость отслеживания целей приводит к понятию контрольных точек – моментов разработки, когда осуществляется подведение промежуточных итогов, осмысление достигнутого и ревизия сделанных ранее предположений.

Из сказанного следует, что контрольные точки являются постоянной заботой менеджера проекта и моментами, когда интенсивность его работы возрастает. Вместе с тем определение контрольных точек- это элемент планирования, который находится в компетенции менеджера. В первую очередь планирования времени, а на базе его – распределения остальных ресурсов. Имеется определенная свобода в выборе этапов и контрольных точек, ограниченная обязательствами перед заказчиками, разработчиками, а также планировщиками компании. Это означает целесообразность приспособления этапов развития проекта к его специфике и к специфике условий выполнения задания.

Понятие жизненного цикла занимает центральное место в методологиях программирования. Оно образует базу для естественной систематизации инструментов и методов, ресурсов и результатов на разных этапах разработки и использования программных систем. Понятие это не является специфическим для программирования. Оно возникло и развивалось сначала применительно к техническим системам. В частности, еще недавно наши экономисты выражали беспокойство по поводу того, что зарубежный потребитель

сравнительно дешевым советским тракторам предпочитает канадские, цена которых в несколько раз выше. Оказалось, что полная стоимость последних с учетом затрат всего "жизненного цикла существования машин" (включая их техническое обслуживание и ремонт) получается в конечном счете в несколько раз меньше. Не случайно вопрос технологичности с точки зрения не только изготовления, но и последующей эксплуатации имеет в технике первостепенное значение.

Понятие жизненного цикла программного обеспечения появилось, когда программистское сообщество осознало необходимость перехода от кустарных ремесленнических методов разработки программ к более технологичному мануфактурному, а в перспективе и к промышленному, их производству. Особенность программной индустрии заключается в том, что сотрудник, соответствующий в традиционной схеме мануфактурного производства неквалифицированному рабочему, должен иметь квалификацию и работать на уровне как минимум техника, а квалифицированный рабочий – уже на том уровне, который в технике соответствует инженеру. Как обычно происходит в подобных ситуациях, программисты прежде всего попытались перенести опыт других индустриальных производств в свою сферу. В частности, было заимствовано и модифицировано под реальный опыт программирования понятие жизненного цикла технической системы.

Аналогия жизненного цикла программного обеспечения с техническими системами имеет и более глубокие корни, и более фундаментальные различия, чем это может показаться на первый взгляд. Программы, в отличие от чаще всего встречающихся в нашем обиходе искусственных объектов, или артефактов, являются в некотором роде идеальными объектами и на самом деле единственными чисто искусственными объектами, кроме математических конструкций, с которыми имеет дело человек. Например, машина сделана из реальных материалов, наследует их свойства и уже по этой причине не может создаваться чисто логически, силами одной лишь мысли. А математический объект и программа состоят из информационных сущностей. В принципе, они могут быть созданы чисто логически. Но и в том и в другом случае чистая логика творения натыкается на реальные либо конвенциональные ограничения. Математический объект должен быть признан сообществом математиков и поэтому должен вписаться в систему существующих математических объектов. Программа же создается на базе других программ и должна работать в их окружении. Сложность программного окружения такова, что разобраться в нем до конца невозможно, да оно вдобавок все время меняется. Так что программное окружение играет сейчас для программ ту же роль, что конструкционные материалы и окружающая среда для технических систем.

И, конечно же, неустраним фактор пользователя. Все равно, делаете вы программу для конечных пользователей либо для квалифицированных программистов, пользователь перепутает все, что возможно, и даже то, что невозможно, и трудно предсказать, что он может сотворить с программой. Но тем не менее программа наиболее близко, за исключением математических структур, подходит к понятию настоящего искусственного объекта. Программы не подвержены физическому износу, но в ходе их эксплуатации обнаруживаются ошибки (неисправности), требующие исправления.

Ошибки возникают также от изменения условий использования программы. Последнее является принципиальным свойством программного обеспечения, иначе оно теряет смысл. Поэтому правомерно говорить о старении программ, правда не о физическом, а о «моральном». Необходимость внесения изменений в действующие программы (как из-за обнаруживаемых ошибок, так и по причине развития требований) приводит, по сути дела, к тому, что разработка программного обеспечения продолжается после передачи его пользователю и в течение всего времени жизни программ. Деятельность, связанная с решением довольно многочисленных задач такой продолжающейся разработки, получила название сопровождения программного обеспечения

Первоначально понятие жизненного цикла рассматривалось как цикл разработки. Однако понимание того, что стоимость программного обеспечения включает издержки в течение всего времени жизни системы, а не только затраты на разработку или исполнение программ, привело к естественной трансформации исходного понятия цикла разработки. Жизненный цикл – это проекция пользовательского понятия «время жизни» на понятие разработчика «технологический цикл (цикл разработки)». Комбинацией этих понятий объясняется происхождение самого термина «жизненный цикл программного обеспечения».

Каждый проект разработки ПО имеет свой собственный жизненный цикл, состоящий из четырех фаз: инициация, планирование, реализация и завершение.

1.4.1 Инициация проекта

Этап инициации во многом определяет успешность проекта: недостаточное внимание именно к этой фазе проекта неизбежно приводит к существенным проблемам при планировании, реализации и завершении проекта.

Процессы инициации связаны с организационными процессам и поэтому часто выполняются вне рамок проекта. В ходе процесса инициации уточняются первоначальное описание содержания, планируемые ресурсы, документируются исходные допущения и ограничения.

Для каждого проекта должна быть сформулирована Концепция. Если проект небольшой, то для изложения концепции достаточно несколько абзацев.

Цель данного документа – подтверждение и согласование единого видения целей, задач и результатов всеми участниками проекта. Концепция проекта используется для принятия решений в ходе всего проекта, а также на фазе приемки – для подтверждения результата.

Как правило, концепция проекта содержит следующие разделы:

1) название проекта;
2) цели проекта;
3) результаты проекта;
4) допущения и ограничения;
5) ключевые участники и заинтересованные стороны;
6) ресурсы проекта;
7) сроки;
8) риски;
9) критерии приемки;
10) обоснование целесообразности проекта.

1.4.2 Планирование проекта

Уточнение содержания и состава работ

Согласно теории систем, наиболее эффективным способом решения сложной задачи является анализ и декомпозиция задачи на более простые подзадачи, которые, в свою очередь, могут быть разделены на функции и так далее. При этом деление осуществляется с соблюдением иерархии. Таким образом, формируется некоторая иерархическая структура или

дерево, в корне которого находится проект, а на листьях элементарные задачи или работы, которые надо выполнить, чтобы завершить проект в условиях заданных ограничений.

Согласно терминологии PMBOK, иерархическая структура работ (ИСР, Work /Breakdown Structure, WBS) – ориентированная на результат иерархическая декомпозиция работ, выполняемых командой проекта для достижения целей проекта и необходимых результатов. С ее помощью структурируется и определяется все содержание проекта. Каждый следующий уровень иерархии отражает более детальное определение элементов проекта.

Основой для разработки ИСР служит концепция проекта.

Планирование управления содержанием

Если на начальном этапе работы над проектом ИС заказчик думает, что точно знает, что хочет, то, как правило, его требования изменяются («плывут») в ходе выполнения проекта. Бороться с данной проблемой практически бесполезно. Для минимизации последствий рекомендуется следующее: сразу, как только удалось стабилизировать и согласовать ИСР, необходимо разработать план управления содержанием проекта.

Планирование организационной структуры

Поскольку проект представляет собой «живой» организм, то и организационная структура может динамично меняться в ходе проекта. Организационная структура в обязательном порядке должна включать в себя систему рабочих взаимоотношений между рабочими группами проекта, систему отчетности, оценки хода выполнения проекта и систему принятия решений.

Планирование управления конфигурациями

План проекта должен включать в себя работы по обеспечению единого хранилища всей проектной документации и разрабатываемого программного кода, обеспечению сохранности и восстановление проектной информации после сбоя, работы по настройке рабочих станций и серверов, используемых участниками проектной команды. Кроме этого в плане должны содержаться работы, необходимые для организации сборки промежуточных выпусков системы, а также ее конечного варианта.

Планирование управления качеством

Основная задача обеспечения качества это не поиск ошибок в готовом продукте (выходной контроль) а их предупреждение в процессе производства.

Базовое расписание проекта

После определения трудоемкости работ необходимо составить расписание работ по проекту, т.е. определить график выполнения работ и общие сроки реализации проекта

Базовое расписание – это утвержденный план-график с указанными временными фазами проекта, контрольными точками и элементами иерархической структуры работ.

Для проекта следует определить критический путь проекта (Critical path) – самую длинную цепочку работ в проекте. Увеличение длительности любой работы в этой цепочки приводит к увеличению длительности всего проекта.

1.4.3 Реализация проекта

Рабочее планирование

Для управления проектом требуется осуществлять стратегическое и оперативное планирование. Базовое расписание, составленное на этапе планирования проекта, служит

ориентиром для мониторинга состояния дел на макроуровне. Для оперативного управления проектом используется рабочий план.

Для рабочего планирования целесообразно использовать систему управления задачами или багтрекинга, поскольку она позволяет задавать последовательность переходов задачи от исполнителя к исполнителю, управлять приоритетами работ и адекватно отслеживать их статус: анализ, проектирование, кодирование, тестирование, документирование. Работа должна считаться законченной только тогда, когда реализация требования протестирована и документирована.

Принципы количественного управления

Для осуществления эффективного управления необходимо, чтобы на проекте были определены количественные ключевые показатели. Для каждого измеримого показателя должны быть определены его плановые значения. Для каждого планового значения должны быть определены три области критичности отклонений:

- допустимые отклонения – предполагается, что никаких управляющих воздействий не требуется;
- критичные отклонения – требуется тщательный анализ причин отклонения и при необходимости применение корректирующих действий;
- недопустимые отклонения – требуется срочный анализ причин отклонения и обязательное применение корректирующих действий.

1.4.4 Завершение проекта

Главная цель этой фазы – проверить и передать заказчику результат проекта. Для этого необходимо выполнить приемо-сдаточные работы в соответствии с процедурой приемки.

Результаты проекта должны быть переданы во внедрение или сопровождение, или должным образом законсервированы для дальнейшего использования. Не должно оставаться «зависших» работ по проекту.

Важная задача, которая должна быть решена на данной фазе, это реализация обратной связи по проекту с целью сохранения результатов, знаний и опыта, полученных в проекте для более эффективного и качественного выполнения аналогичных проектов в будущем. Необходимо архивировать все результаты, документировать опыт, уроки по проекту и предложения по улучшению технологии выполнения работ и управления проектами.

Лекция 2. Планирование проекта

2.1 Планирование проекта

Термин **проект**, как известно, происходит от латинского слова **projectus**, что в буквальном переводе означает «*брошенный вперед*». Таким образом, объект управления, который можно представить в виде проекта, отличает возможность его перспективного развертывания, т.е. возможность предусмотреть его состояния в будущем. Хотя различные официальные источники трактуют понятие проекта по-разному, во всех определениях четко просматриваются особенности проекта как объекта управления, обусловленные комплексностью задач и работ, четкой ориентацией этого комплекса на достижение определенных целей и ограничениями по времени, бюджету, материальным и трудовым ресурсам.

Однако, любая деятельность, в том числе и та, которую никто не собирается называть проектом, выполняется в течение определенного периода времени и связана с затратами определенных финансовых, материальных и трудовых ресурсов. Кроме того, любая разумная деятельность, как правило, целесообразна, т.е. направлена на достижение определенного результата. И, тем не менее, в одних случаях к управлению деятельностью подходят как к управлению проектом, а в других случаях – нет.

Деятельность как объект управления рассматривается в виде проекта тогда, когда

- она объективно имеет комплексных характер и для ее эффективного управления важное значение имеет анализ внутренней структуры всего комплекса работ (операций, процедур и т.п.);
- переходы от одной работы к другой определяют основное содержание всей деятельности;
- достижение целей деятельности связано с последовательно-параллельным выполнением всех элементов этой деятельности;
- ограничения по времени, финансовым, материальным и трудовым ресурсам имеют особое значение в процессе выполнения комплекса работ;
- продолжительность и стоимость деятельности явно зависит от организации всего комплекса работ.

Поэтому, **объектом проектного управления** принято считать *особым образом организованный комплекс работ, направленный на решение определенной задачи или достижение определенной цели, выполнение которого ограничено во времени, а также связано с потреблением конкретных финансовых, материальных и трудовых ресурсов*. При этом под «работой» понимается элементарная, неделимая часть данного комплекса действий.

Элементарность работы – понятие условное и относительное. То, что нецелесообразно делить в одной системе действий, полезно разукрупнять в другой. Например, если за элемент комплекса работ по сборке автомобиля принимается технологическая операция, то одной из «работ» может считаться установка сборщиком фары. Эта «работа» в данном случае неделима, так как остаются неизменными ее факторы – исполнитель, предмет и объект действия. Но, как только мы начинаем рассматривать исполнение этой работы как отдельную задачу, она сама превращается в комплекс.

На Рисунок 1 в обобщенной форме показаны основные этапы выполнения проектирования. На первом этапе определяются отдельные процессы, составляющие проект, их отношения предшествования (т.е. какой процесс должен предшествовать другому) и их длительность. Далее проект представляется в виде сети, показывающей отношения

предшествования среди процессов, составляющих проект. На третьем этапе на основе построенной сети выполняются вычисления, в результате которых составляется временной график реализации проекта.

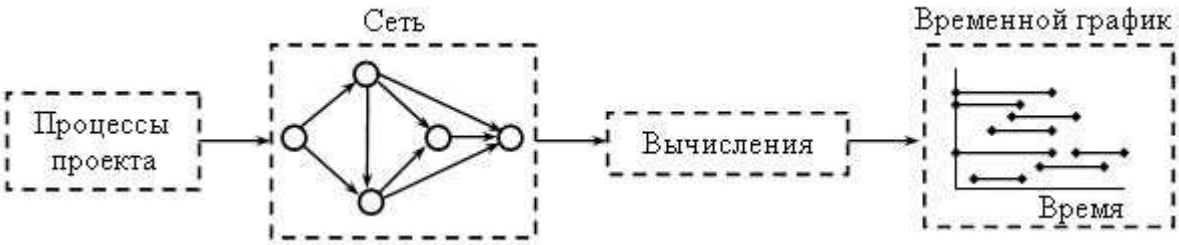


Рисунок 1. Основные этапы выполнения проектирования

2.2 Календарно-сетевое планирование

Одним из первых методов оптимизации хода выполнения работ, не потерявшим своей эффективности и по сей день, является методика диаграмм Гантта. Она был разработана Генри Ганттом для отслеживания хода строительства больших трансконтинентальных океанских лайнеров. Идея Гантта состояла в том, что главным ресурсом планирования является время, а основой принятия управленческих решений – сравнение запланированного и фактического состояния работ. На диаграммах Гантта по горизонтали обычно показывают интервалы времени, а по вертикали – работы, операции, оборудование. Горизонтальные отрезки отражают длительность выполнения работ. Выбрав по горизонтальной оси текущий момент времени и получив оперативную информацию о ходе производства, можно сопоставить фактическое состояние дел и планировавшееся. Все современные системы управления проектами и планирования предлагают представление графиков работ в виде диаграмм Гантта.

В то же время диаграммы Гантта имеют ряд недостатков. Например, с их помощью довольно сложно планировать многовариантные взаимосвязанные цепочки работ (в строительных, военных, государственных проектах и на производстве). Для таких задач в военном ведомстве США в 1950-е годы были предложены методы сетевого планирования, или методы выбора "критического пути". Кроме того, диаграммы Гантта удобно применять только для одного критического ресурса – времени. При необходимости учитывать еще несколько ресурсов, например технологическую оснастку, диаграммы Гантта надо воспринимать как объемные, приобретающие ряд измерений по числу учитываемых ресурсов. Это имеет смысл для визуальной интерпретации планов, но затрудняет их анализ.

Современные методы управления проектами уходят корнями в 50-е годы текущего столетия. Практически одновременно две проектные группы представили методы управления сложными комплексами работ.

В 1956 году М. Уолкер из фирмы DuPont, исследуя возможности более эффективного использования принадлежащей фирме вычислительной машины Univac, объединил свои усилия с Д. Келли из группы планирования капитального строительства фирмы «Ремингтон Рэнд». Они попытались использовать ЭВМ для составления планов-графиков крупных комплексов работ по модернизации заводов фирмы DuPont. В результате был создан рациональный метод описания проекта с использованием ЭВМ. Первоначально он получил название метода Уолкера-Келли, а позже переименован в метод критического пути (Critical Path Method – CPM).

Данный метод имеет достоинства:

- позволяет получить графическое представление проекта,
- определяет ориентировочное время, требуемое для его выполнения, и

– показывает, какие действия критичны, а какие не столь важны для соблюдения всего графика работ.

Методика, предлагаемая CPM, в настоящее время широко распространена, однако она имеет свои недостатки. Оптимизации методом CPM поддаются только сравнительно легко понятные проекты, в которых не трудно спрогнозировать время выполнения действия. Поэтому при разработке или конструировании различных систем (когда одна из интеллектуальных задач может быть не решаема достаточно долгие время) CPM применим лишь условно.

Для задач, связанных с интеллектуальным трудом и другими вопросами, в которых стоимость оптимизируемого параметра не известна наверняка, используется метод PERT-анализа (Program Evaluation Review Technique). Он был разработан сотрудниками Военно-морского флота США в 1957 году для обеспечения создания ракеты "Поларис". Его разработали корпорация Lockheed Air Craft, консалтинговая компания Booz, Allen & Hamilton и особое проектное бюро ВМС США в процессе создания ракетного комплекса Polaris. На его разработку, по заявлениям фирмы, ушло 15 лет, таким образом, начало работ относилось к 1943г. Благодаря PERT проект, который состоял из 60 тыс. операций и объединял около 3800 основных подрядчиков, удалось закончить на два года раньше запланированного срока. Его успешное завершение способствовало тому, что вскоре данный метод стал повсеместно применяться для планирования проектов в вооруженных силах США. Применяя PERT-анализ, они попытались сымитировать график выполнения работ по созданию ракеты путем построения логической сети взаимозависимых последовательных событий. На начальной стадии PERT-представление было сфокусировано на контроле временных характеристик графика и прогнозировании вероятности успешного завершения программы. Но прежде чем PERT-представление было окончательно принято руководителями программ в промышленности, Военно-воздушные силы США внесли дополнение в методику, добавив к логической сети функцию ресурсной оценки. Таким образом, в 1962 году появилась PERT/Cost-методика (PERT-анализ с целью стоимостного прогнозирования), в то время как первоначально PERT-анализ был известен под названием PERT/Time (PERT-анализ для определения времени реализации проекта).

Использование метода PERT позволило руководству программы точно знать, что требуется делать в каждый момент времени и кто именно должен это делать, а также какова вероятность своевременного завершения отдельных операций. Руководство программой оказалось настолько успешным, что проект удалось завершить на два года раньше запланированного срока. Благодаря такому впечатляющему началу, данный метод управления вскоре стал использоваться для планирования проектов во всех вооруженных силах США. Методика отлично себя зарекомендовала при координации работ, выполняемых различными подрядчиками в рамках крупных проектов по разработке новых видов вооружения.

На практике метод позволяет оценить предполагаемое время окончания проекта, вероятность его завершения к конкретной дате и конкретизировать действия, наиболее неопределенные в смысле своего выполнения.

Идеи, сходные с идеями, положенными в основу системы PERT, были еще в 30-х годах предложены в советском капитальном строительстве (на строительстве Магнитогорского металлургического комбината), но в то время они не получили распространения и для них не были произведены необходимые математические разработки.

Однако это не означает, что в нашей стране идеи метода никого не интересовали. Благодаря усилиям С.П. Никанорова, в 60-е годы Министерство обороны в лице подведомственных институтов активно занялось разработками в этой области.

Оба метода были основаны на использовании сетевых диаграмм, но CPM оперировал только одной длительностью работы, в то время как PERT учитывал четыре длительности --

оптимистическую, пессимистическую, наиболее вероятную и средневзвешенную. Это обусловлено различными сферами применения методов.

PERT появился при выполнении проекта, окружающая среда которого характеризовалась высокой степенью неопределенности, поэтому приходилось оценивать разные варианты завершения работ. Степень неопределенности проектной среды, в которой создан метод CPM, была существенно меньше, и исполнители могли довольно точно оценить длительность работ, основываясь на предыдущем опыте.

За прошедшее время произошла взаимная интеграция методов, и сейчас при планировании в основном используется Метод критического пути.

Методы **CPM** и **PERT** отличаются тем, что в методе критического пути длительность каждого этапа проекта является ***детерминированной***, тогда как в системе планирования **PERT** - ***стохастической***.

2.3 Методы планирования и управления проектами и ресурсами

Для описания, анализа и оптимизации проектов наиболее подходящими оказались сетевые модели, представляющие из себя разновидность ориентированных графов. Сетевые графики способствуют определению направленных критических работ, оценивают резервы напряженных работ, планируют сроки выполнения всего комплекса работ. Сетевые графики строятся с помощью двух элементов: работ и событий. Под работой понимается некоторое неделимое действие, характеризующееся затратами ресурсов и временем исполнения. Под событием понимается начало или окончание какой-либо работы (одной или нескольких). Операция – это сама работа или действие.

Одно событие может иметь несколько предшествующих и/или последующих событий. Если наступлению данного события не предшествует какая-либо работа, то это событие называется **исходным**. Событие, не имеющее последующих работ, называется **завершающим**. Любая последовательность работ в сети, в которой конечное событие каждой работы последовательности совпадает с начальным событием следующей за ней работы, называется путем. Следует различать два вида пути:

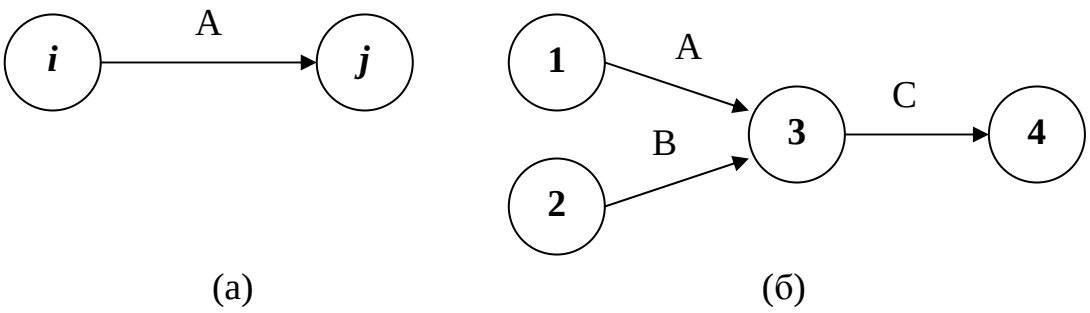
- полным путем называется непрерывная последовательность выполнения работ от исходного до завершающего события;
- критическим путем называется путь от исходного до завершающего события, который характеризуется наибольшей продолжительностью выполнения работ, находящихся на этом пути.

В сетевой модели роль вершин графа могут играть события, определяющие начало и окончание отдельных работ, а дуги в этом случае будут соответствовать работам. Такую сетевую модель принято называть *сетевой моделью с работами на дугах или моделью «дуга – работа»* (Activities on Arrows, **AoA**). В то же время, возможно, что в сетевой модели роль вершин графа играют работы, а дуги отображают соответствие между окончанием одной работы и началом другой. Такую сетевую модель принято называть *сетевой моделью с работами в узлах или моделью «узел – работа»* (Activities on Nodes, **AoN**).

2.3.1 Модель «дуга – работа»

Сетевой график проекта раскрывает его внутренние связи, служит основой для календарного планирования работ и использования оборудования, облегчает взаимодействие менеджеров и исполнителей.

Сетевая модель отображает взаимосвязи между операциями (работами, задачами) и порядок их выполнения (отношение упорядочение или следования). Для представления операции используется стрелка (ориентированная дуга), направление которой соответствует процессу реализации проекта во времени. Отношение упорядочения между операциями задается с помощью событий. Событие определяется как момент времени, когда завершаются одни операции и начинаются другие. Начальная и конечная точки любой операции описываются парой событий, которые называют начальным событием и конечным событием. Операции, выходящие из некоторого события, не могут начаться, пока не будут завершены все операции, входящие в это событие. По принятой терминологии каждая операция представляется ориентированной дугой, а каждое событие – узлом (вершиной).



(а) (б)

Рисунок 2

На Рисунок 2 (а) приведен пример графического изображения операции **A** с начальным событием *i* и конечным *j*. На Рисунок 2 (б) показан другой пример, из которого видно, что для возможности начала операции **C** требуется завершение операций **A** и **B**. Протекание операций во времени задается путем нумерации событий, причем номер начального события всегда меньше номера конечного.

При построении сетевого графика должны выполняться следующие требования:

1) В каждой последовательности работ очередное событие не может наступить, если не наступило предшествующее.

2) Каждая работа изображается в виде только одной стрелки, т.е. два события соединяются не более чем одной стрелкой.

На рисунке показано, как с помощью фиктивного процесса можно представить два параллельных (конкурирующих) процесса **A** и **B**. По определению фиктивный процесс (который на схеме сети обычно обозначается пунктирной дугой) не поглощает временных или других ресурсов. Вставив фиктивный процесс одним из четырех способов, показанных на Рисунок 3, мы получаем возможность идентифицировать процессы **A** и **B**, по крайней мере, одним уникальным концевым узлом (как требует правило 2).

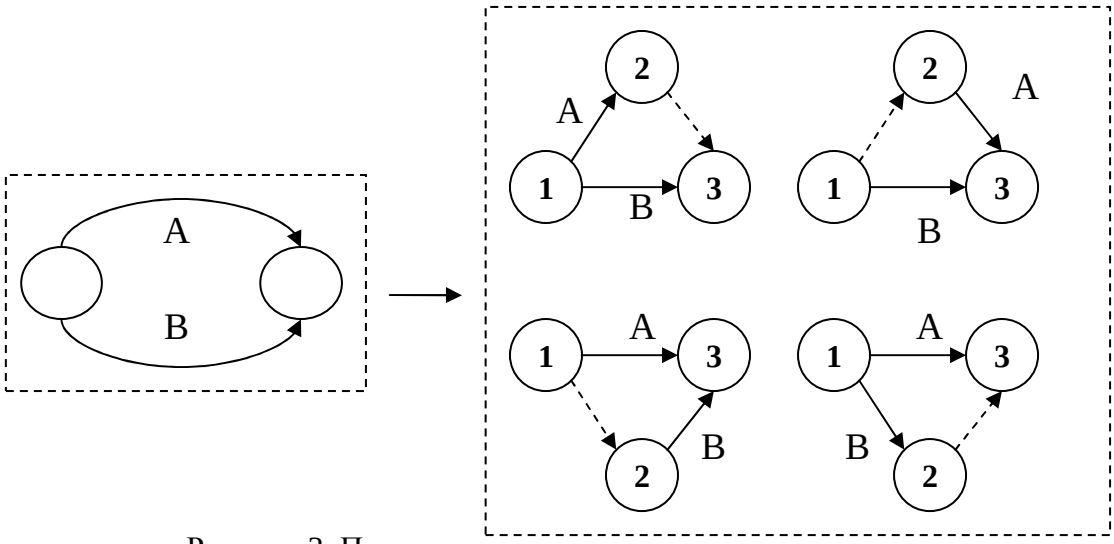


Рисунок 3. Представление конкурирующих процессов

3) На сетевом графике должно быть одно событие, в которое не входят никакие работы – исходное событие (начало работ), и одно событие, из которого не выходят работы – завершающее событие (конец работ).

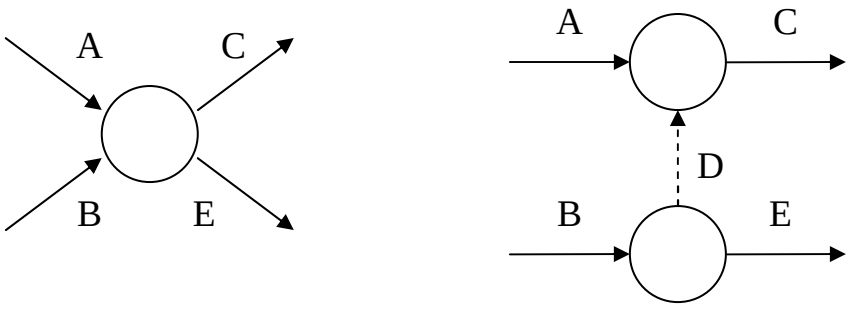
4) На сетевом графике не должно быть стрелок, которые ниоткуда не выходят и никуда не входят. Все события, кроме исходного и завершающего, должны иметь как входящие, так и выходящие стрелки.

5) Если появляются события, отличные от исходного и завершающего, в которых нет входящих и выходящих стрелок, то они соединяются с началом или концом сетевого графика фиктивными работами, продолжительность которых равна нулю.

Предположим, например, что четыре процесса должны удовлетворять следующим условиям.

1) Процесс **C** должен начаться после завершения процессов **A** и **B**.
2) Процесс **E** должен начаться непосредственно после завершения процесса **B**.

На Рисунок 4 (а) показано неправильное представление наших процессов, так как из него следует, что процесс **E** должен начаться после завершения как процесса **B**, так и **A**. На рисунке (б) показано, как с помощью фиктивного процесса **D** разрешить эту коллизию.



(а) (б)

Рисунок 4. Представление процессов

Пример 1.

Табл. 1

Операция	Непосредственно предшествующие	Операция	Непосредственно предшествующие
A	нет	G	F
B	A	H	D
C	B	I	D
D	B	J	H, I
E	D	K	G, J
F	C, E	L	K

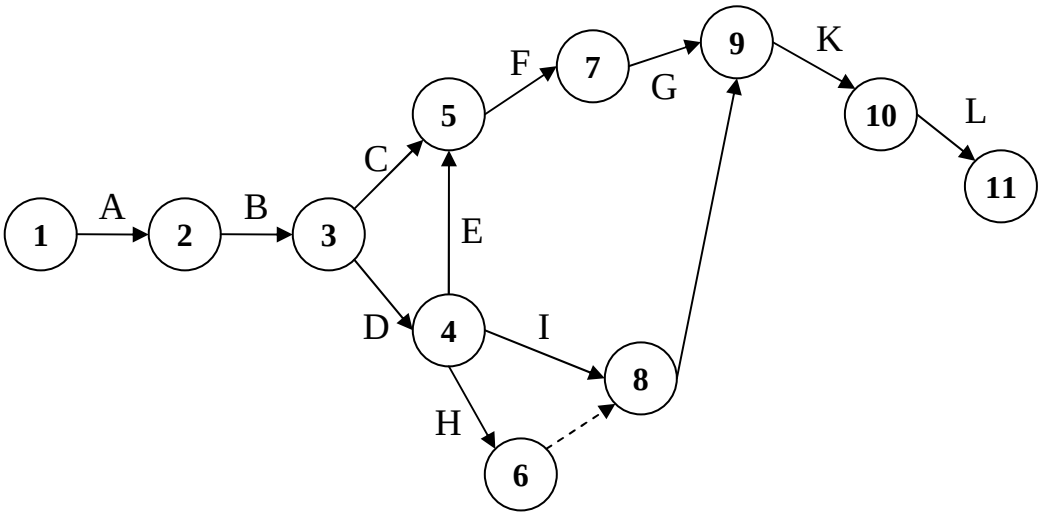


Рисунок 5. Сетевая модель для примера 1

2.3.2 Расчет сетевой модели

Построение сети является лишь первым шагом на пути к получению календарного плана, определяющего сроки начала и окончания каждой операции. Вследствие наличия взаимосвязей между различными операциями для определения сроков их начала и окончания необходимо проведение специальных расчетов. Эти расчеты можно выполнять непосредственно на сети, пользуясь простыми правилами. В результате вычислений определяются критические и некритические операции проекта. Операция считается **критической**, если задержка ее начала приводит к увеличению срока окончания всего проекта. **Некритическая** операция отличается тем, что промежуток времени между ее ранним началом и поздним окончанием (в рамках рассматриваемого проекта) больше ее фактической продолжительности. В таком случае говорят, что некритическая операция имеет резерв, или запас времени.

2.3.3 Определение критического пути

Критический путь определяет непрерывную последовательность критических операций, связывающих начальное и завершающее события сети. Другими словами, критический путь задает все критические операции проекта. Метод определения такого пути проиллюстрируем на следующем примере.

Графически **события изображаются кружками**, разделенными на три равных сегмента (радиусами под углом в 120°); **работы изображаются сплошными линиями со**

стрелками на конце, ориентированными слева направо; **фиктивные работы изображаются пунктирными линиями со стрелками на конце**, ориентированными слева направо.

Пример 2. Рассмотрим сетевую модель, показанную на Рисунок 6, с исходным событием 0 и завершающим событием 6. Оценки времени, необходимого для выполнения каждой операции и обозначения операций, даны у стрелок.

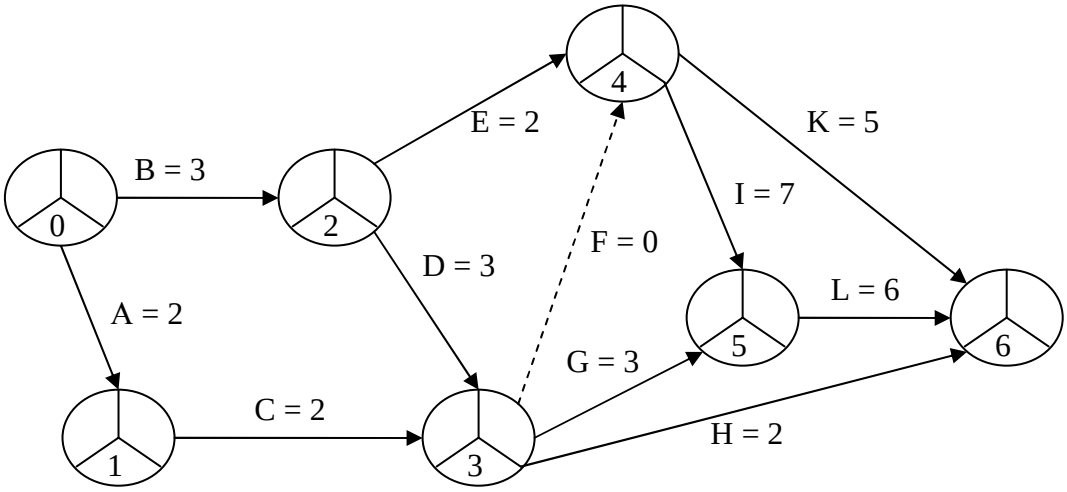


Рисунок 6

Расчет критического пути включает два этапа. Первый этап называется прямым проходом. Вычисления начинаются с начального события и продолжаются до тех пор, пока не будет достигнуто завершающее событие всей сети. Для каждого события *j* вычисляется одно число ES_j, представляющее ранний срок его наступления (ранний срок окончания всех операций, входящих в событие *j*; ранний срок начала всех операций, выходящих из события *j*).

На втором этапе, называемом обратным проходом, вычисления начинаются с завершающего события сети и продолжаются, пока не будет достигнуто начальное событие. Для каждого события *i* вычисляется число LF_i, представляющее поздний срок его наступления (поздний срок окончания всех операций, входящих в событие *i*, поздний срок начала всех операций, выходящих из события *i*).

Первый этап

Если принять $i = 0$, т.е. считать, что номер исходного события сети равен нулю, то при расчете сети полагаем $ES_0 = 0$. Обозначим символом D_{ij} (Duration) продолжительность операции (i, j). Тогда вычисления при прямом проходе выполняются по формуле $ES_j = \max_i \{ES_i + D_{ij}\}$, где max берется по всем операциям, завершающимся *j*-ом событии. Следовательно, чтобы вычислить ES_j для события *j*, нужно сначала определить ES_i начальных событий ***всех*** операций (i, j), входящих в событие *j*.

Применительно к Рисунок 6 вычисления начинаются с $ES_0 = 0$. Далее получим:

$$ES_1 = ES_0 + D_{01} = 0 + 2 = 2, \quad ES_2 = ES_0 + D_{02} = 0 + 3 = 3,$$

$$ES_3 = \max_{i=1,2}\{ES_i + D_{i3}\} = \max\{2 + 2; 3 + 3\} = 6,$$

$$ES_4 = \max_{i=2,3}\{ES_i + D_{i4}\} = \max\{3 + 2; 6 + 0\} = 6,$$

$$ES_5 = \max_{i=3,4}\{ES_i + D_{i5}\} = \max\{6+3;6+7\} = 13,$$

$$ES_6 = \max_{i=3,4,5}\{ES_i + D_{i6}\} = \max\{6+2;6+5;13+6\} = 19.$$

На этом вычисления первого этапа заканчиваются.

Второй этап начинается с завершающего события сети, для которого полагаем $LF_n = ES_n$, где n – завершающее событие. Затем, для любого события i $LF_i = \min\{LF_j - D_{ij}\}$, где min берется по всем операциям, выходящим из i-го события. Далее получим:

$$LF_6 = ES_6 = 19, \;\; LF_5 = LF_6 - D_{56} = 19 - 6 = 13,$$

$$LF_4 = \min_{j=5,6}\{LF_j - D_{4j}\} = \min\{13-7;19-5\} = 6,$$

$$LF_3 = \min_{j=4,5,6}\{LF_j - D_{3j}\} = \min\{6-0;13-3;19-2\} = 6,$$

$$LF_2 = \min_{j=3,4}\{LF_j - D_{2j}\} = \min\{6-3;6-2\} = 3,$$

$$LF_1 = LF_3 - D_{13} = 6 - 2 = 4,$$

$$LF_0 = \min_{j=1,2}\{LF_j - D_{0j}\} = \min\{4-2;3-3\} = 0.$$

Таким образом, вычисления при обратном проходе закончены.

Теперь, используя результаты вычислений первого и второго этапа, можно определить операции критического пути. Операция (i, j) принадлежит критическому пути, если она удовлетворяет следующим трем условиям:

$$ES_i = LF_i, \qquad (1)$$

$$ES_j = LF_j, \qquad (2)$$

$$ES_j - ES_i = LF_j - LF_i = D_{ij}. \qquad (3)$$

По существу, эти условия означают, что между ранним сроком начала (окончания) и поздним сроком начала (окончания) критической операции запас времени отсутствует. В сетевой модели это отражается в том, что для критических операций числа, проставленные у начальных и конечных событий, совпадают, а разность между числом у конечного события и числом у начального события равна продолжительности соответствующей операции.

На Рисунок 7 критический путь включает операции {**B, D, F, I, L**}. Критический путь определяет кратчайшую возможность всего проекта в целом. Заметим, что операции (2, 4), (3, 5), (3, 6) и (4, 6) удовлетворяют условиям (1) и (2), но не условию (3). Поэтому они не являются критическими. Отметим также, что критический путь представляет собой непрерывную цепочку операций, соединяющую исходное событие сети с завершающим.

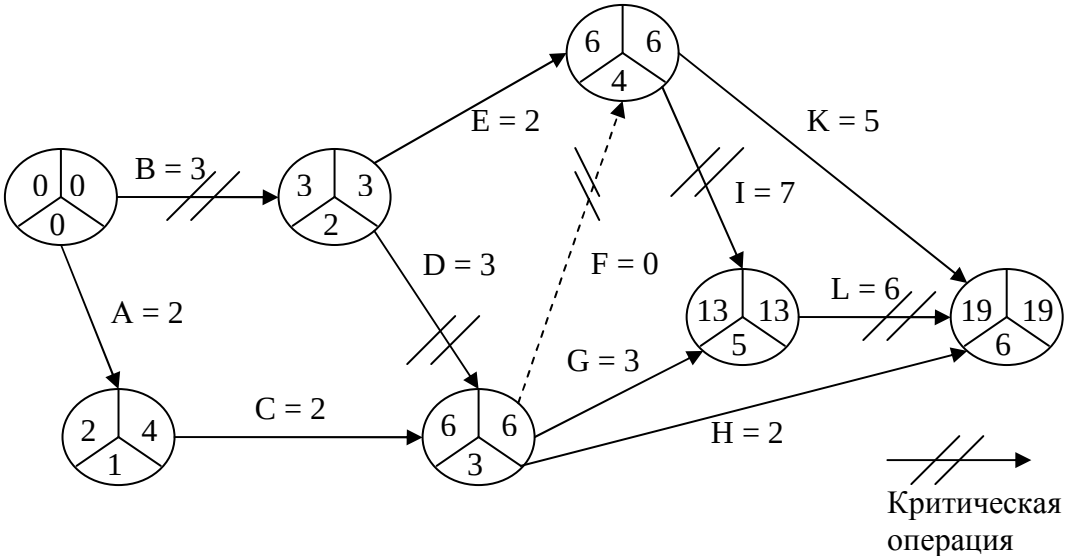


Рисунок 7

2.3.4 Определение резервов времени

После определения критического пути необходимо вычислить резервы времени для некритических операций. Очевидно, что резерв времени для критической операции должен быть равен нулю. Поэтому она и называется критической. Рассмотрим произвольную операцию (i, j).

Наиболее ранний возможный срок начала операции (i, j) – ES_{ij} (Early Start) – определяется при допущении, $ES_{ij} = ES_i$, поскольку работа не может начаться раньше наступления предшествующего события i. Отсюда следует, что наиболее ранний возможный срок окончания операции (i, j) (Early Finish): $EF_{ij} = ES_{ij} + D_{ij}$.

Наиболее поздний допустимый срок окончания работы (i, j) – LF_{ij} (Late Finish) – определяется как самое позднее время завершения работы без задержки срока окончания всего проекта. Поскольку операция должна быть закончена не позднее наибольшего допустимого срока наступления последующего события j, то имеем $LF_{ij} = LF_j$. Отсюда следует, что наиболее поздний допустимый срок начала работы (i, j) – LS_{ij} (Late Start) вычисляется следующим образом: $LS_{ij} = LF_{ij} - D_{ij}$.

Резерв времени является показателем гибкости планирования сроков некритических работ в сетевой модели. Можно определить четыре показателя: **полный**, **свободный**, **независимый** и **гарантированный** резервы времени.

Полный резерв времени TF_{ij} (Total Float) для работы (i, j) представляет собой максимальную продолжительность задержки работы (i, j), не вызывающую задержки в осуществлении всего проекта. Он вычисляется как $TF_{ij} = LS_{ij} - ES_{ij} = LF_{ij} - EF_{ij}$.

Свободный резерв времени FF_{ij} (Free Float) для работы (i, j) является показателем максимальной задержки работы (i, j), не влияющей на начало последующих работ. Операции со свободным резервом уникальны, так как выполнение операции может откладываться, не влияя на ранний старт следующих операций. Изменение сроков операции со свободным резервом требует меньше координации с другими участками проекта. Он вычисляется как $FF_{ij} = ES_{ij} - EF_{ij}$.

Независимый резерв времени IF_{ij}. Не оказывает никакого влияния на предшествующие и последующие операции. Независимый резерв времени является удобным показателем свободы планирования сроков. Он представляет собой максимальную продолжительность задержки работы (i, j) без задержки последующих работ, если все предшествующие работы заканчиваются как можно позже, т.е. $IF_{ij} = \max\{0, ES_j - (LF_i + D_{ij})\}$.

Гарантированный резерв времени SF_{ij} – это максимально возможная задержка работы, не влияющая на окончательный срок выполнения проекта, если предшествующие работы выполняются с запаздыванием.

$SF_{ij} = LF_{ij} - (LF_i + D_{ij})$.

Результаты расчета критического пути и резервов времени некритических операций для нашего примера можно свести в удобную для использования Табл. 2.

Табл. 2

Операция (i, j)	D_{ij}	Раннее		Позднее		Полный резерв $TF_{ij} = LS_{ij} - ES_{ij}$	Свободный резерв $FF_{ij} = ES_{ij} - EF_{ij}$	Независимый резерв IF_{ij}	Гарантированный резерв SF_{ij}
		начало ES_{ij}	окончание EF_{ij}	начало LS_{ij}	окончание LF_{ij}				
A (0,1)	2	0	2	2	4	2	0	0	2
B (0,2)	3	0	3	0	3	0	0	0	0
C (1,3)	2	2	4	4	6	2	2	0	0
D (2,3)	3	3	6	3	6	0	0	0	0
E (2,4)	2	3	5	4	6	1	1	1	1
F (3,4)	0	6	6	6	6	0	0	0	0
G (3,5)	3	6	9	10	13	4	4	4	4
H (3,6)	2	6	8	17	19	11	11	11	11
I (4,5)	7	6	13	6	13	0	0	0	0
K (4,6)	5	6	11	14	19	8	8	8	8

L (5,6)	6	13	19	13	19	0	0	0	0

Таблица содержит всю необходимую для построения календарного плана (графика) информацию. Заметим, что только критические операции должны иметь нулевой *полный* резерв времени. Когда полный резерв равен нулю, свободный резерв также должен быть равен нулю. Однако обратное неверно, поскольку свободный резерв некритической операции также может быть нулевым. Так, например, в Табл. 2 свободный резерв времени некритической операции (0,1) равен нулю.

Замечание 1. Необходимо учитывать тот факт, что при вычислении полного резерва времени принимается неявное допущение, согласно которому все предшествующие работы (во всяком случае, те, которые имеют какое-либо отношение к рассматриваемой работе) должны выполняться как можно раньше, чтобы обеспечить полный резерв времени для данной работы. Следовательно, в общем случае практически невозможно для каждой работы реализовать собственный полный резерв времени.

Замечание 2. Свободный резерв времени для определенной работы не может превышать полный резерв.

Замечание 3. Различные показатели резерва времени помогают распределять имеющиеся ресурсы для каждой работы. При наличии резерва времени имеется некоторая свобода распределения ресурсов.

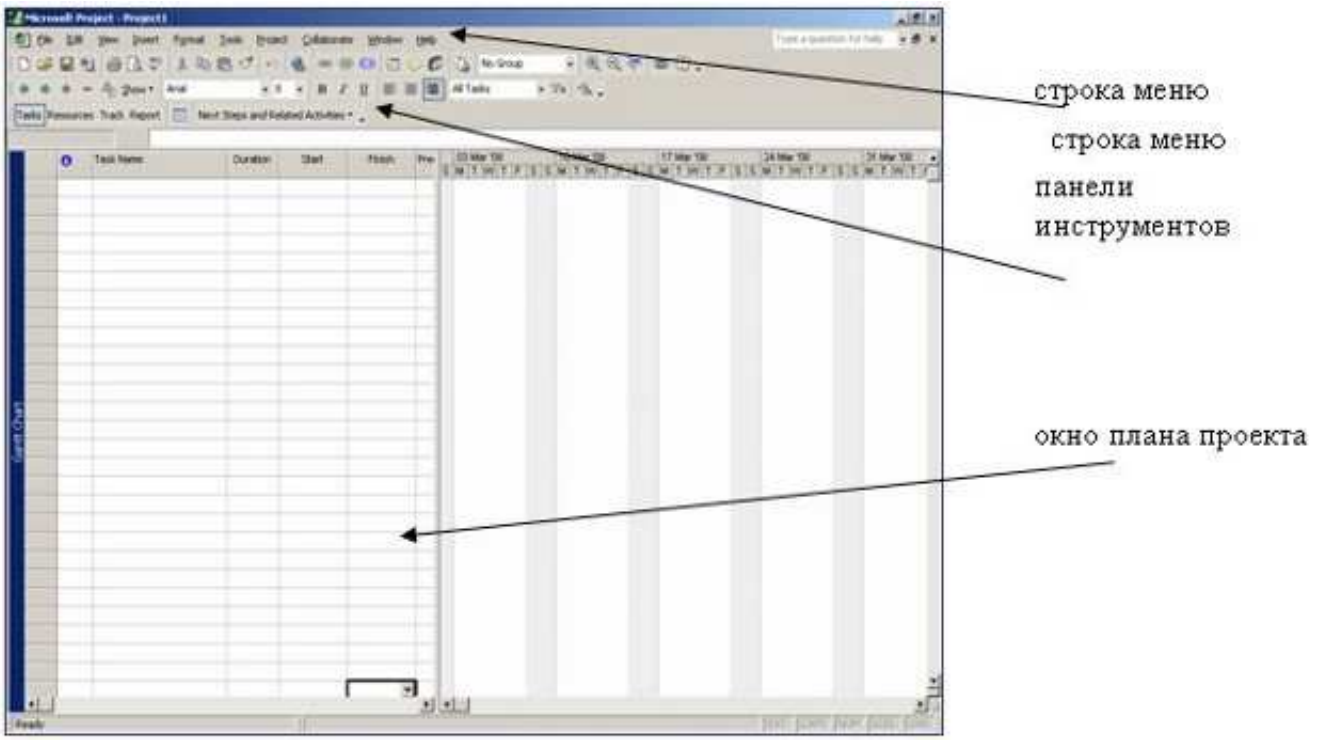
2.4 Среда MS Project

Является на сегодняшний день самой распространенной в мире системой управления проектами. Этот пакет используют для планирования своих проектов около 3 миллионов людей. Его стандартный офисный интерфейс позволяет быстро научиться использовать продукт. Во многих западных компаниях MS Project стал привычной добавкой к Microsoft Office даже для рядовых сотрудников, которые используют его для планирования графиков несложных комплексов работ.

MS Project является мощным инструментом, помогающим управлять процессом планирования и выполнения проекта. Работа в среде MS Project требует от пользователей знакомства с теорией управления проектами, знания основных терминов и понятий. Общепринятые методики управления проектами, стандарты и ключевые термины содержатся в руководстве по управлению проектами " Project Management Body of Knowledge" (PMBOK), которое, по существу, является сводом профессиональных знаний по управлению проектами.

Процесс планирования в среде MS Project имеет свою специфику. После того как определены цели проекта, сформулированы основные этапы, определено содержание основных этапов, создается план проекта; далее необходимо ввести и структурировать список задач проекта, для каждой задачи ввести длительность, установить зависимости между задачами; затем, создать список ресурсов: сотрудников, оборудования и материалов, назначить ресурсы на задачи. На основе введенной информации MS Project создает расписание. Созданное таким образом расписание можно настраивать и оптимизировать.

Основные элементы интерфейса программы MS Project представлены на рисунке.



В MS Project имеется несколько режимов отображения проектной информации - **представлений (view)**, они отображают проектную информацию в различных аспектах. По умолчанию, при запуске MS Project план проекта отображается в представлении **Gantt Chart (Диаграмма Ганта)**. Перейти из одного представления в другое можно используя меню **View (Вид)** или специальную панель **View Bar (Панель представлений)**, где перечислены все возможные представления.

Лекция 3. Планирование и контроль проектных работ

3.1 Планирование и контроль проектных работ

Перечислим перечень основных задач, для решения которых используются системы управления проектами (СУП):

- разработку расписания исполнения проекта без учета ограниченности ресурсов;
- разработку расписания исполнения проекта с учетом ограниченности ресурсов (leveling);
- определение критического пути и резервов времени исполнения операций проекта;
- определение потребности проекта в финансировании, материалах и оборудовании;
- определение распределения во времени загрузки возобновляемых ресурсов;
- анализ рисков и планирование расписания с учетом рисков;
- учет исполнения проекта;
- анализ отклонений хода работ от запланированного и прогнозирование основных параметров проекта.

3.2 Программные средства для управления ИТ-проектами: Open Plan, Spider Project, Primavera

В настоящее время на российском рынке представлено разнообразное программное обеспечение по управлению проектами (более 200 программных продуктов и сотни фирм разработчиков), помогающее специалистам в различных сферах бизнеса, например:

1. MS Project (компания – разработчик Microsoft)
2. Time Line (компания – разработчик Time Line Solutions)
3. Primavera (компания – разработчик Primavera Systems, Inc.)
4. Spider Project (компания – разработчик Spider Project)
5. Artemis Views (компания – разработчик Artemis Internationals)
6. Open Plan (компания – разработчик WELCOM)

Рассмотрим основные возможности некоторых из этих программ.

1) Time Line (Time Line Solutions)

Очень многие компании в нашей стране, в том числе и строительные, начинали свой путь к внедрению систем управления проектами именно с этого продукта. Этот пакет начал продаваться еще в начале девяностых. Были локализованы две версии - 5.0 для DOS и 1.0 для Windows. Отличная функциональность и при этом простота использования, сделали его весьма распространенным пакетом. Очень хорошей, по тем временам, была возможность создание вычисляемых пользовательских полей. В дистрибутив пакета входит генератор отчетов Crystal Report.

В 1995 году, уже под лейблом **Symantec,** была выпущена версия 6.5 для Windows. Недостатком этой версии можно считать не очень хорошо реализованный принцип WYSIWYG. На этом развитие пакета, к сожалению, остановилось. Локализированной версии выпущено не было. По сведениям компании, занимавшейся продвижением **Time Line** на российском рынке, продажи его прекращены около 2 лет назад.

Time Line 1.0, подобно MS Project, содержит лишь минимально необходимые функции управления проектами, предоставляя пользователю-непрофессионалу максимально простые и ясные средства быстрого создания и расчета несложных проектов.

Начинающему пользователю система предлагает набор базовых расписаний, дающих общее представление о проектах в различных областях (бизнес-план, производство изделия, маркетинг изделия, новостройка и т.д.). Специальная функция Инструктор активизирует модуль контроля за логикой работы пользователя. Периодически он выводит на экран запросы, уточняющие назначение проделанных операций, и предложения относительно дальнейших действий.

Пакет содержит полный набор функций управления проектами, однако, объем планируемых проектов, как и в MS Project, ограничен 10000 задач и 1000 видов ресурсов. Система предоставляет упрощенные алгоритмы ресурсного планирования.

Средства создания отчетов кроме табличных и графических (Гантт, PERT) позволяют получать календарный график, который представляет данные в хорошо знакомом руководителю формате настенного календаря. Использование правил отбора позволяет напечатать индивидуальный рабочий календарь для групп сотрудников или каждого из сотрудников в отдельности. Данное средство может быть удобно для небольших проектов.

Для организации коллективной работы с данными проекта TimeLine 1.0 может быть установлен как на рабочих станциях, так и на сервере сети.

Многопроектное управление реализуется только через объединение проектов или связь проектов. Пакет поддерживает импорт/экспорт данных в форматах ASCII, CSV, Lotus 1-2-3, dBase. В комплект поставки русской версии TimeLine 1.0 входят дополнительные продукты Guide Line и Guide Line Maker, предназначенные для создания и использования инструкций по разработке проектов в конкретных предметных областях.

TimeLine 1.0 может быть рекомендован пользователям-непрофессионалам, планирующим преимущественно временные и стоимостные параметры проектов.

TimeLine 6.5 является более мощной версией системы управления проектами, принципиально отличающейся от версии 1.0 по ряду параметров.

Основными отличительными особенностями TimeLine 6.5 являются реализация концепции многопроектного планирования в рамках организации, гибкие средства поддержки формирования отчетов и средства настройки на пользовательскую информационную среду. В TimeLine 6.5 сняты ограничения на размерность проектов.

TimeLine 6.5 позволяет хранить все данные, касающиеся проектов организации в единой SQL базе данных, которая, кроме описания проектов и единого для организации списка ресурсов, содержит все элементы настроек управленческой среды принятой в компании для работы с проектами. Все основные объекты базы данных объединены в окне OverView в соответствующих разделах. С помощью данного окна можно просмотреть структуру базы данных проекта и осуществить доступ к любому элементу, а также создать свои пользовательские элементы в списках.

TimeLine 6.5 предлагает достаточно мощные алгоритмы работы с ресурсами, включающие средства межпроектного назначения и выравнивания перегрузок ресурсов, гибкие возможности по описанию специфических календарных графиков работы ресурсов. Недостатком данных средств является отсутствие возможностей описания и отображения иерархии ресурсов организации.

Стандартные возможности генерации табличных отчетов по проекту дополнены возможностями включаемой в поставку TimeLine 6.5 системы создания и генерации отчетов Cristal Reports 4, которая позволяет создавать практически любые виды отчетов, содержащие

данные как из БД TimeLine, так и из других баз данных компании. Более 30 заготовок стандартных отчетов управления проектами в формате Cristal Reports включены в систему.

Полезной дополнительной возможностью системы являются средства создания собственных формул в электронной таблице TimeLine.

Отдельный модуль импорта/экспорта позволяет обмениваться данными с другими пакетами УП (MS Project, CA-SuperProject, Time Line 1.0 for Windows и 5.0 для DOS), базами данных (dBase) и электронными таблицами (Lotus). TimeLine 6.5 поддерживает стандарты ODBC, OLE 2.0, DDE, поддерживает макроязык Symantec Basic.

TimeLine 6.5 может быть рекомендован для планирования средних или комплексов малых проектов.

2) Primavera (Primavera Systems, Inc.)

Центральный программный продукт семейства Primavera, Primavera Project Planner (P3), хорошо известен в среде профессиональных менеджеров проектов во всем мире. Сегодня P3 (цена для американского рынка - $4000) применяется для управления средними и крупными проектами в самых различных областях, хотя наибольшее распространение данный продукт получил в сфере управления строительными и инженерными проектами.

Primavera Project Planner предоставляет достаточно стандартный для всех подобных систем графический интерфейс, но у P3 есть несколько дополнительных возможностей. Во-первых, это возможность группировки и упорядочивания работ по различным признакам на разных уровнях детализации проекта, что позволяет представить информацию в более удобном виде для конкретной управленческой ситуации. Например, используя данные средства, всю информацию по проекту можно сгруппировать по фазе проекта на первом уровне иерархии, по ответственному ресурсу на втором и отсортировать по дате начала работ на третьем. Для каждой группы могут быть заданы собственные шрифт и цвет (текста и фона), постраничное разбиение.

Другая полезная особенность – возможность разбиения экрана по горизонтали на две части, каждая из которых может быть просмотрена независимо. Это дает возможность одновременно просматривать разные части проекта.

Кроме того, P3 имеет определенные отличия от других пакетов в средствах ресурсного планирования. При описании ресурса могут быть указаны нормальное и максимальное количество наличия данного ресурса, а также его цена по шести временным интервалам. Ресурс может быть помечен как управляющий (объем назначения управляющего ресурса на задачу будет влиять на длительность ее выполнения). Например, определив, что рабочие - это управляющий ресурс, а бригадир - нет, можно добиться сокращения сроков выполнения задачи прокладка траншеи за счет назначения большего количества рабочих. Увеличение же количества бригадиров не повлияет на длительность работы.

При планировании загрузки ресурсов может возникнуть необходимость в описании нелинейного профиля потребления ресурса отдельной задачей. P3 дает возможность описать различные кривые распределения ресурса, предлагая девять стандартных кривых и возможность определить собственный профиль потребления, разбив временную фазу задачи на 10 периодов.

Средства автоматического перепланирования задач с учетом ограничений на ресурсы приобретают особую важность для крупных проектов, когда менеджер не в состоянии самостоятельно проанализировать причины нехватки ресурсов и найти решение для каждой конкретной работы. P3 позволяет выбрать режим перерасчета расписания и подобрать критерий перепланирования работ, обеспечивающий получение более короткого расписания. Среди режимов перерасчета можно выделить выравнивание вперед (определение возможной даты окончания проекта при заданной начальной дате), выравнивание назад (определение самой

поздней допустимой даты начала проекта), сглаживание перегрузок ресурсов в рамках в временных резервов работ или в рамках заданного интервала. Кроме того, имеется возможность перераспределять назначение работ между сгруппированными ресурсами.

К недостаткам средств ресурсного планирования можно отнести ограничение на количество календарей. Кроме главного календаря проекта, P3 позволяет описать лишь 30 дополнительных календарей, в то время как возможность задания индивидуальных графиков работы для каждого ресурса уже стало нормой в современных пакетах управления проектами). Другое ограничение связано с количеством ресурсов (не более 120), контролируемых при выравнивании профиля загрузки ограниченных ресурсов.

Средства поддержки многопроектной среды управления в P3 включают возможность определения иерархии и права доступа к мастер-проекту и подпроектам. Менеджер-координатор проекта имеет право редактировать мастер-проект и все подпроекты. Менеджер подпроекта имеет право добавлять ресурсы в словарь ресурсов, но не удалять их и не изменять их цены. Если разрешение ресурсных конфликтов в рамках подпроекта требует данные другого подпроекта, менеджер может это сделать только при наделении его дополнительными полномочиями со стороны менеджера-координатора проекта. Однако, ресурсное планирование по всему проекту в целом может осуществляться только менеджером-координатором. Только он может определить связи между подпроектами. По сравнению со многими другими программными продуктами, которые также дают возможность многопроектного управления, отличительной особенностью P3 является подробное описание принципов многопроектного управления в документации, где они рассматриваются с двух точек зрения: менеджера-координатора проекта и менеджера подпроекта (хотя считается, что тема мультипроектного управления требует дополнительного учебника).

Кроме P3, компанией Primavera Systems поставляется облегченная система для управления проектами - SureTrak ($700). Этот программный продукт ориентирован на небольшие проекты, подпроекты, работу конкретных исполнителей с фрагментами проектов. SureTrak имеет те же средства, что и P3 в плане организации проекта по кодам и фильтрации информации, установки ограничений и расчета расписания, но в то же время существует ряд ограничений и дополнительных возможностей.

Из ограничений следует отметить отсутствие средств многопроектного управления и фрагментации проектов, меньшую размерность проектов, более скромные средства создания отчетов. Однако, в SureTrak появились календари ресурсов и, как следствие возможность расчета длительностей работ с учетом согласования календарей исполнителей (ожидается, что календари ресурсов появятся и в следующей версии P3). Кроме того, у ресурсов появилась дополнительная категория - доход. SureTrak отличается от всех остальных продуктов Primavera тем, что он полностью русифицирован и поставляется вместе с руководством для пользователя на русском языке.

SureTrak осуществляет импорт/экспорт файлов в форматах P3 и MS Project.

Таким образом, работая совместно, P3 и SureTrak предлагают масштабируемый подход к управлению проектами различного размера и сложности. Кроме вышеназванных продуктов из семейства Primavera интерес может представлять система анализа рисков проекта Monte Carlo for Primavera.

3) Spider Project

Spider Project (разработчик/представитель в России – компания “Технологии управления "Спайдер", www.spiderproject.ru)

Без преувеличения можно сказать, что **Spider Project** лучшая *отечественная* система управления проектами. Версия под DOS появилась еще в 1992 году. От версии к версии заметно улучшается не только интерфейс системы, но и ее функциональность.

По информации, полученной от специалистов, разрабатывающих и поддерживающих пакет (Spider Technologies Group), система была инсталлирована для управления несколькими десятками крупных проектов.

Данный пакет имеет несколько отличительных особенностей, позволяющих ему конкурировать с западными системами на крупных, промышленных проектах.

Во-первых, это мощные алгоритмы планирования использования ограниченных ресурсов. Тестирование известных пакетов управления проектов показало преимущество алгоритмов Spider Project по качеству составляемых планов выполнения работ при ограниченности имеющихся ресурсов. Для 32 из 100 проектов, участвовавших в тестировании, Spider Project составил более короткие расписания работ, а для остальных 68 его расписания не уступали лучшим из расписаний, составленных западными пакетами.

В пакете реализована возможность использования при составлении расписания работ взаимозаменяемых ресурсов (пулы ресурсов), которая также позволяет получить более короткие расписания. Использование ресурсных пулов избавляет менеджера от необходимости жестко назначать исполнителей на работы проекта. Ему достаточно указать общее количество необходимых для производства работ ресурсов и из каких ресурсов это количество выбирать. Это позволяет и сократить непроизводительные простои ресурсов и облегчить работу проектного менеджера, избавляя его от необходимости производить утомительные на больших проектах оценки "что если".

Еще одной особенностью пакета является возможность использование нормативно-справочной информации - о производительностях ресурсов на тех или иных видах работ, расходе материалов, стоимостях работ и ресурсов. Spider Project позволяет неограниченно наращивать число учитываемых в проектах показателей, создавать и использовать в расчетах любые дополнительные табличные документы и базы данных, вводить любые формулы расчета. Возможность настройки системы позволяет пользователям получать от пакета не только расписание работ, графики загрузки ресурсов и стоимостные характеристики проекта, но и технологические характеристики составленных расписаний. Так, например, в горно-добывающей промышленности пользователи Spider Project получили возможность планировать не только порядок выемки объемов руды, но и учитывать объемы отдельных компонентов, содержащихся в руде.

Превосходя многие западные пакеты по мощности и гибкости отдельных функций, Spider Project, в целом, уступает в области программной реализации (использование стандартов обмена данными, пользовательский интерфейс и т.д.). На сегодняшний день не завершен полный перевод системы в среду Windows. Пакет имеет Windows надстройку, ввод и отображение данных в диаграммах Гантт и PERT, однако программы расчета по-прежнему функционируют в DOS. Для создания пользовательских табличных отчетов по проекту необходимо использовать программу электронных таблиц AUTOPLAN (DOS версия), которая входит в поставку Spider Project.

Программа поддерживает традиционные подходы к управлению проектами, описанные в PMBOK Guide и реализованные в большинстве западных пакетов, но при этом предлагает то, что не имеет аналогов ни в теории, ни в практике управления проектами на Западе. Методология, положенная в основу Spider Project, естественна для российских менеджеров, но не поддерживается зарубежными программами. Spider Project с успехом применяется в Голландии, поэтому можно считать, что подходы, используемые в проекте, не определяются чисто Российской спецификой.

3.3 Средства управления версиями SVN

3.4 Средства Task Tracking и Bug Tracking

Лекция 4. Модели разработки ПО

4.1 Модели жизненного цикла ПО

Моделью жизненного цикла информационной системы будем называть некоторую структуру, определяющую последовательность осуществления процессов, действий и задач, выполняемых на протяжении жизненного цикла информационной системы, а также взаимосвязи между этими процессами, действиями и задачами.

В стандарте ISO/IEC 12207 не конкретизируются в деталях методы выполнения действий и решения задач, входящих в процессы жизненного цикла информационной системы, а лишь описываются структуры этих процессов. Это вполне понятно, так как регламенты стандарта являются общими для любых моделей жизненного цикла, методологий и технологий разработки. Модель же жизненного цикла зависит от специфики информационной системы и условий, в которых она создается и функционирует.

К настоящему времени наибольшее распространение получили две основные (классические) модели жизненного цикла:

- каскадная модель, иногда также называемая моделью водопада (waterfall);
- спиральная модель.

4.1.1 Каскадная модель жизненного цикла

Каскадная модель предусматривает последовательную организацию работ. При этом основной особенностью является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как полностью завершены все работы на предыдущем этапе. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Можно выделить следующий ряд устойчивых этапов разработки, практически не зависящих от предметной области (рис. 1):

Рисунок 1. Каскадная модель разработки

Результатом, получаемым на первом этапе, является техническое задание (задание на разработку), согласованное со всеми заинтересованными сторонами.

Результатом второго этапа является комплект проектной документации, содержащей все необходимые данные для реализации проекта.

Результатом выполнения третьего этапа является готовый программный продукт.

Результатом выполнения четвертого этапа являются различного рода скрытые недостатки, проявляющиеся в реальных условиях работы информационной системы.

Главная задача пятого этапа – убедить заказчика, что все его требования выполнены в полной мере.

В действительности жизненный цикл самой системы существенно сложнее и длиннее. Он может включать в себя произвольное число циклов уточнения, изменения и дополнения уже принятых и реализованных проектных решений. В этих циклах происходит развитие информационной системы и модернизация отдельных ее компонентов.

Основные достоинства каскадной модели

- На каждом этапе формируется законченный набор проектной документации, отвечающей критериям полноты и согласованности. На заключительных этапах также разрабатывается пользовательская документация, охватывающая все предусмотренные стандартами виды обеспечения информационной системы (организационное, методическое, информационное, программное, аппаратное).
- Выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при разработке определенных информационных систем. Имеются в виду системы, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу выбора реализации, наилучшей с технической точки зрения. К таким информационным системам, в частности, относятся сложные расчетные системы, системы реального времени.

Недостатки каскадной модели

Перечень недостатков каскадной модели при ее использовании для разработки информационных систем достаточно обширен:

- существенная задержка в получении результатов;
- ошибки и недоработки на любом из этапов проявляются, как правило, на последующих этапах работ, что приводит к необходимости возврата назад;
- сложность параллельного ведения работ по проекту;
- чрезмерная информационная перенасыщенность каждого из этапов;
- сложность управления проектом;
- высокий уровень риска и ненадежность инвестиций.

Задержка в получении результатов обычно считается главным недостатком каскадной схемы. Данный недостаток проявляется в основном в том, что из-за последовательного подхода к разработке согласование результатов с заинтересованными сторонами производится только после завершения очередного этапа работ.

Используемые при разработке информационной системы модели автоматизируемого объекта, отвечающие критериям внутренней согласованности и полноты, могут в силу различных причин устареть за время разработки (например, из-за внесения изменений в законодательство, колебания курса валют и т.п.). Это относится и к функциональной модели, и к информационной модели, и к проектам интерфейса пользователя, и к пользовательской документации.

Возврат на более ранние стадии. Данный недостаток каскадной модели, в общем-то, является одним из проявлений предыдущего. Возврат может служить причиной срыва графика работ и усложнения взаимоотношений между группами разработчиков, выполняющих отдельные этапы работы.

Самым же неприятным является то, что недоработки предыдущего этапа могут обнаруживаться не сразу на последующем этапе, а позднее (например, на стадии опытной эксплуатации могут проявиться ошибки в описании предметной области). Это означает, что часть проекта должна быть возвращена на начальный этап работы. Вообще, работа может быть возвращена с любого этапа на любой предыдущий этап, поэтому в реальности каскадная схема разработки выглядит так, как показано на рис. 2.

Рисунок 2. Реальный процесс разработки по каскадной схеме

Сложность параллельного ведения работ. Проблемы возникают вследствие того, что работа над проектом строится в виде цепочки последовательных шагов. Причем даже в том случае, когда разработку некоторых частей проекта (подсистем) можно вести параллельно, при использовании каскадной схемы распараллеливание работ весьма затруднительно. Сложности параллельного ведения работ связаны с необходимостью постоянного согласования различных частей проекта. Чем сильнее взаимозависимость отдельных частей проекта, тем чаще и тщательнее должна выполняться синхронизация, тем сильнее зависят друг от друга группы разработчиков.

Информационная перенасыщенность. Проблема информационной перенасыщенности возникает вследствие сильной зависимости между различными группами разработчиков. Данная проблема заключается в том, что при внесении изменений в одну из частей проекта необходимо оповещать всех разработчиков, которые использовали или могли бы использовать эту часть в своей работе. Как следствие, объем документации по мере разработки проекта растет очень быстро, так что требуется все больше времени для составления документации и ознакомления с ней.

Следует также отметить, что, помимо изучения нового материала, не отпадает необходимость в изучении старой информации. Это связано с тем, что вполне вероятна ситуация, когда в процессе разработки изменяется состав группы разработчиков (этот процесс носит название **ротации кадров**).

Сложность управления проектом при использовании каскадной схемы в основном обусловлена строгой последовательностью стадий разработки и наличием сложных взаимосвязей между различными частями проекта.

Последовательность разработки проекта приводит к тому, что одни группы разработчиков должны ожидать результатов работы других команд. Поэтому требуется административное вмешательство для согласования сроков работы и состава передаваемой документации.

Высокий уровень риска. Чем сложнее проект, тем больше продолжительность каждого из этапов разработки и тем сложнее взаимосвязи между отдельными частями проекта, количество которых также увеличивается. Возврат на предыдущие стадии может быть связан не только с ошибками, но и с изменениями, произошедшими в предметной области или в требованиях заказчика за время разработки. Причем возврат проекта на доработку вследствие этих причин не гарантирует, что предметная область снова не изменится к тому моменту, когда будет готова следующая версия проекта. Фактически это означает, что существует вероятность того, что процесс разработки «зациклится» и система никогда не дойдет до сдачи в эксплуатацию.

Поэтому можно утверждать, что сложные проекты, разрабатываемые по каскадной схеме, имеют повышенный уровень риска.

4.1.2 Спиральная модель жизненного цикла

Спиральная модель, в отличие от каскадной, предполагает итерационный процесс разработки информационной системы. При этом возрастает значение начальных этапов жизненного цикла, таких как анализ и проектирование. На этих этапах проверяется и обосновывается реализуемость технических решений путем создания прототипов.

Итерации

Каждая итерация представляет собой законченный цикл разработки, приводящий к выпуску внутренней или внешней версии изделия (или подмножества конечного продукта), которое совершенствуется от итерации к итерации, чтобы стать закопченной системой (рис. 3).

Таким образом, каждый виток спирали соответствует созданию фрагмента или версии программного изделия, на нем уточняются цели и характеристики проекта, определяется его качество, планируются работы на следующем витке спирали. На каждой итерации углубляются и последовательно конкретизируются детали проекта, в результате чего выбирается обоснованный вариант, который доводится до окончательной реализации.

Использование спиральной модели позволяет осуществлять переход на следующий этап выполнения проекта, не дожидаясь полного завершения текущего — недоделанную работу можно будет выполнить на следующей итерации. Главная задача каждой итерации — как можно быстрее создать работоспособный продукт, который можно показать пользователям системы. Таким образом, существенно упрощается процесс внесения уточнений и дополнений в проект.

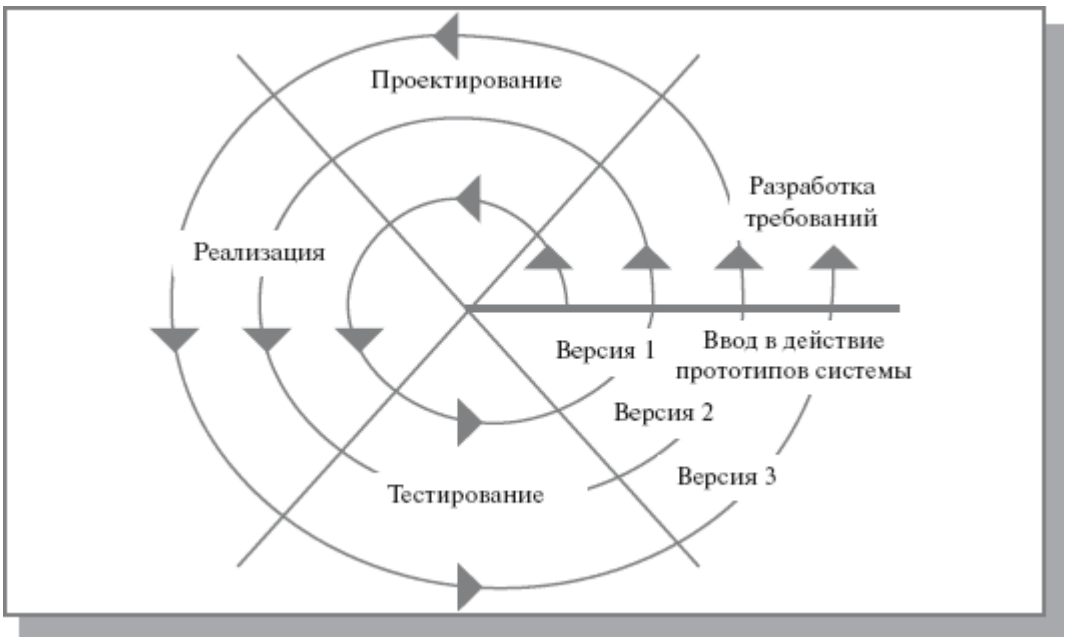


Рисунок 3. Спиральная модель жизненного цикла информационной системы

Преимущества спиральной модели

Спиральный подход к разработке программного обеспечения позволяет преодолеть большинство недостатков каскадной модели и, кроме того, обеспечивает ряд дополнительных возможностей, делая процесс разработки более гибким.

Недостатки спиральной модели

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Иначе процесс разработки может превратиться в бесконечное совершенствование уже сделанного. При итерационном подходе полезно следовать принципу «лучшее – враг хорошего». Поэтому завершение итерации должно производиться строго в соответствии с планом, даже если не вся запланированная работа закончена.

4.2 Жесткие и гибкие методологии

Контроль деятельности проекта – задача, требующая специальных организационных подходов. Общей целью является превращение создания программного продукта в упорядоченный процесс, в рамках которого развитие проекта можно сделать более прогнозируемым и эффективным. Для этого обычно создается детальное описание процесса разработки системы, особое место в котором занимает планирование. Иначе говоря, создается метод, с помощью которого предполагается конструировать систему. Удачные методы обобщаются, в результате опыт их применения превращается в методологию. Нередко при таком формировании методологии фиксируются частные и случайные решения, оказавшиеся полезными из-за специфики удачных проектов. Наряду со всем полезным эти решения объявляются обязательными, стандартными, их вынуждены применять и тогда, когда исходные предпосылки утрачены. Чтобы следовать таким методологиям, приходится выполнять множество различных предписаний, что замедляет темп основных программистских работ. Поэтому их называют тяжеловесными, жесткими или, монументальными.

Жесткие методологии привлекательны для заказчиков программных проектов, которые всегда могут проверить, действительно ли процесс разработки упорядочен и результаты соответствуют планам.

В настоящее время разработаны стандарты зрелости процессов разработки программного обеспечения в организациях. Среди них наиболее развитым является предложение Института программной инженерии при Университете Карнеги–Меллона – так называемая модель SW-CMM (Capability Maturity Model for Software). Эту модель можно считать общепринятой, поскольку на нее чаще всего ориентируются заказчики, в частности из Министерства обороны США, предлагая проекты только тем организациям, которые сертифицированы на уровнях зрелости 4 и 5. Модель CMM была сформирована при существенном влиянии практики военных заказов с их жесткой процедурой контроля и отчетности. Предложения CMM для определения зрелости организации опираются на то, какие процедуры управления программными проектами, отслеживания их развития и другие менеджерские методы приняты в качестве фирменного стандарта. Методологии программирования, построенные на базе этих предложений, часто рассматривают как эталон жесткости.

В противовес жестким методологиям в последнее время сформировался компромиссный подход, методологии которого объединены общим термином agile development. На русский язык его переводят как быстрое развитие, гибкие методологии и даже «шустрые технологии». В этом подходе пытаются предоставить возможность облегченной организованной работы программистских коллективов, когда перегруженность стандартизованного процесса препятствует эффективности. Они претендуют на то, что их применение возможно даже когда не удается достаточно точно представить проект при его зарождении, т.е. в довольно типичной ситуации неопределенности реальной пользовательской потребности. В этих случаях предлагается привлечение заказчика к формированию задач и корректировке предположений в течение всего развития проекта. С его помощью пытаются выявлять наиболее точно и без излишних бюрократических процедур актуальные потребности пользователей, сложившиеся на текущий момент. В результате появляется возможность указать именно те требования, реализация которых необходима и допускает максимально короткие сроки выпуска релиза.

Все методологии быстрого развития ориентируются на стратегию итеративного наращивания возможностей системы, но с частичным отказом от постулата неизменности априорной архитектуры (как следствие, не только допускается, но даже предполагается, что архитектура системы, а значит, и программный код будут меняться при переходе от релиза к релизу). Все они предоставляют разработчикам значительно большую свободу, чем, к примеру, требования стандартов CMM. Но не следует думать, что этот подход полностью отменяет жесткие методологии. Необходим баланс между свободой быстрых методологий и дисциплиной.

Более детально охарактеризовать все методологии быстрого развития совместно не представляется возможным – слишком различны их исходные принципы, слишком зависимы они от специфики коллективов, занимающихся разработками программного обеспечения, от зачастую стихийно складывающихся предпочтений, привычек. На стратегическом уровне их действительно объединяет так называемый манифест живой разработки (Agile Manifesto), принятый энтузиастами в феврале 2001 года в местечке Сноуберд (США), который сводится к четырем постулатам:

1) индивидуумы и их взаимодействие важнее процессов и инструментов;
2) работоспособное программное обеспечение важнее обширной документации;
3) сотрудничество с заказчиком важнее заключения контракта;
4) готовность к изменениям важнее следования плану.

С точки зрения теории деятельности стратегическое разграничение между жесткими и быстрыми методологиями связано с характером видов деятельности, на поддержку которых нацелен каждый из подходов.

Для жестких методологий характерно стремление обеспечить разработчиков рецептами, не требующими обсуждений: нужно выполнять известные предписания, соблюдать регламенты, чтобы соответствующие операционные маршруты приводили к допустимым траекториям. Иными словами, жесткие методологии поддерживают преимущественно такие деятельности, которые можно назвать императивными. В этой поддержке большой удельный вес имеет ориентация на методы-предписания, которым подчиняются средства и инструменты как элементы деятельности. Поэтому здесь значительное место уделяется планированию, измерениям процесса с целью его отслеживания, менеджменту как деятельности, обеспечивающей не только коррекцию недопустимых траекторий, но и принятие решений в недетерминированных случаях, т.е. тогда, когда затруднен выбор подходящего рецепта для разработчиков.

В отличие от традиционных подходов быстрые методологии ориентируются на то, что деятельность по производству программного обеспечения по сути своей является преимущественно креативной, т.е. такой, в которой от разработчиков требуется не только распознавание ситуаций и применение в них известных методов, но и конструирование новых методов действия.

В Таблица 1 представлено сопоставление жесткой и быстрой стратегий в методологиях программирования.

Таблица 1. Сопоставление жесткой и гибкой стратегий в методологиях программирования

Жесткие методологии	**Гибкие методологии**
Ориентация на предсказуемые процессы разработки программного обеспечения с четко обозначенными целями	Осознание того, что процессы разработки программного обеспечения в принципе непредсказуемы
Распознавание ситуаций и применение готовых методов	Распознавание ситуаций и конструирование методов для работы в них
Планирование, в котором определяются этапы с объемом работ, ресурсами, сроками и уровнем качества работ	Соблюдение баланса между параметрами проекта: объем работ, ресурсы, сроки и уровень качества работ
Заказчик — внешний по отношению к проекту субъект, влияющий на разработку только через предоставление ресурсов и контроль результатов, в том числе по поэтапным срокам выполнения проекта	Заказчик (его представитель) — член команды разработчиков, наделенный правом влиять на разработку; его главной целью является отслеживание актуальности решаемых задач
Ролевое разделение труда работников проекта	Совместная деятельность сотрудников и деперсонифицированная ответственность
Дисциплина и подчинение	Самодисциплина и сотрудничество
Обезличенный процесс, исполнители которого определяются только по квалификационным требованиям	Процесс, максимально учитывающий личностные качества исполнителей

Из этого сопоставления видно, что гибкий процесс больше, чем жесткий, соответствует проектам, в которых требуется в полной мере использовать творческий потенциал сотрудников. При следовании жестким методологиям среди прочих показателей ценности сотрудника существенное место занимают способности выполнять предписания, исполнительность, тогда как при быстром подходе – инициативность, стремление к взаимопомощи. В жестком проекте заказчик противопоставлен исполнителям (одна из функций менеджера непосредственно связана с обеспечением взаимодействия с заказчиком), а необходимым условием быстрого проекта является тесное сотрудничество с заказчиком как с членом команды исполнителей.

4.3 ГОСТ серии 34

В нашей стране госты серий 19 и 34 часто применяются при создании программ и автоматизированных систем, особенно, когда в качестве заказчиков выступают государственные или крупные коммерческие организации.

Стандарты ГОСТ серии 34 обладают следующими достоинствами:

- приемлемый уровень разумности и исполнимости;
- широкая распространенность по сравнению с другими стандартами, привычность терминологии и базовых понятий большинству заказчиков
- Минимальный набор жестких требований и возможность адаптации требований стандартов под конкретные условия тех или иных проектов.

Из сравнительно большого спектра ГОСТов серии 34 обычно применяются следующие:

- ГОСТ 34.003-90 «Автоматизированные системы. Термины и определения»;
- ГОСТ 34.201-89«Виды, комплектность и обозначение документов при создании автоматизированных систем»;
- ГОСТ 34.601-90«Автоматизированные системы. Стадии создания»;
- ГОСТ 34.602-89«Техническое задание на создание автоматизированной системы»;
- ГОСТ 34.603-92«Информационная технология. Виды испытаний автоматизированных систем»;
- РД 50-34.698-90«Автоматизированные системы. Требования к содержанию документов».

Несмотря на ряд достоинств, стандарты ГОСТ серии 34 обладают и рядом существенных недостатков.

- Эти ГОСТы не образуют целостную систему, поскольку разработка данной серии ГОСТов была прервана в начале 1990-х годов и с тех пор эти ГОСТы не актуализируются.
- Автоматизированные системы не рассматриваются как инструмент автоматизации бизнес-процессов и, как следствие, в ГОСТах серии 34 не прорабатываются вопросы изменения бизнес-процессов организации, вызванные внедрением автоматизированной системы.
- Очень поверхностно рассматриваются аспекты обслуживания внедренной автоматизированной системы: обучение персонала, выполнение регламентных процедур и т. п.

– Целый ряд ключевых понятий современного управления проектами, таких как риски, программы проектов и портфели проектов, в ГОСТах серии 34 отсутствуют вовсе, а как следствие, их проработка стандартом не предусмотрена.

4.4 Capability Maturity Model for Software - модель зрелости процессов разработки ПО

Capability Maturity Model – модель зрелости возможностей создания программного обеспечения: эволюционная модель развития способности компании разрабатывать программное обеспечение.

В ноябре 1986 года американский институт Software Engineering Institute (SEI) совместно с Mitre Corporation начали разработку обзора зрелости процессов разработки программного обеспечения, который был предназначен для помощи в улучшении их внутренних процессов.

Разработка такого обзора была вызвана запросом американского федерального правительства на предоставление метода оценки субподрядчиков для разработки ПО. Реальная же проблема состояла в неспособности управлять большими проектами. Во многих компаниях проекты выполнялись со значительным опозданием и с превышением запланированного бюджета. Необходимо было найти решение данной проблемы.

В сентябре 1987 года SEI выпустил краткий обзор процессов разработки ПО с описанием их уровней зрелости, а также опросник, предназначавшийся для выявления областей в компании, в которых были необходимы улучшения. Однако, большинство компаний рассматривало данный опросник в качестве готовой модели, вследствие чего через 4 года вопросник был преобразован в реальную модель, Capability Maturity Model for Software (CMM). Первая версия CMM (Version 1.0), вышедшая в 1991 году, в 1992 году была пересмотрена участниками рабочей встречи, в которой принимали участие около 200 специалистов в области ПО, и членами общества разработчиков.

Методология CMM разрабатывалась и развивалась в США как средство, позволяющее выбирать наилучших производителей для выполнения госзаказов по разработке программного обеспечения. Для этого предполагалось создать критерии оценки зрелости ключевых процессов компании-разработчика и определить набор действий, необходимых для их дальнейшего совершенствования. В итоге методология оказалась чрезвычайно полезной для большинства компаний, стремящихся качественно улучшить существующие процессы проектирования, разработки, тестирования программных средств.

CMM де-факто стал именно таким стандартом. Его применение позволяет поставить разработку ПО на промышленную основу, повысить управляемость ключевых процессов и производственную культуру в целом, гарантировать качественную работу и исполнение проектов точно в срок.

Для оценки степени готовности предприятия разрабатывать качественный программный продукт CMM вводит ключевое понятие «зрелость организации» (Maturity). Незрелой считается организация, в которой:

– отсутствует долговременное и проектное планирование;

– процесс разработки программного обеспечения и его ключевые составляющие не идентифицированы;

– реализация процесса зависит от текущих условий, конкретных менеджеров и исполнителей;

– методы и процедуры не стандартизированы и не документированы;

– результат не предопределен реальными критериями, вытекающими из запланированных показателей, применения стандартных технологий и разработанных метрик;

– процесс выработки решений происходит стихийно, на грани искусства.

В этом случае велика вероятность появления неожиданных проблем, превышения бюджета или невыполнения сроков сдачи проекта. В такой компании, как правило, менеджеры и разработчики не управляют процессами – они вынуждены заниматься разрешением текущих и спонтанно возникающих проблем. Отметим, что на данном этапе развития находится большинство российских компаний.

Основные признаки зрелой организации:

1) в компании имеются четко определенные и регламентированные процедуры управления требованиями, планирования проектной деятельности, управления конфигурацией, создания и тестирования программных продуктов, отработанные механизмы управления проектами;

2) эти процедуры постоянно уточняются и совершенствуются;

3) оценки времени сложности и стоимости работ основываются на накопленном опыте, разработанных метриках и количественных показателях, что дает их достаточно точными;

4) актуализированы внешние и созданы внутренние стандарты на ключевые процессы и процедуры;

5) существуют обязательные для всех правила оформления методологической программной и пользовательской документации;

6) технологии незначительно меняются от проекта к проекту на основании стабильных и проверенных походов и методик;

7) максимально используются наработанные в предыдущих проектах организационный и производственный опыт, программные модули, библиотеки программных средств;

8) активно апробируются и внедряются новые технологии, производится оценка их эффективности.

СММ определяет пять уровней технологической зрелости компании, по которым заказчики могут оценивать потенциальных претендентов на получение контракта, а разработчики могут совершенствовать процессы создания ПО.

1) Начальный. Самый примитивный статус организации. Организация способна разрабатывать ПО. Организация не имеет явно осознанного процесса, и качество продукта целиком определяется индивидуальными способностями разработчиков. Один проявляет инициативу и команда следует его указаниям. Успех одного проекта не гарантирует успех другого. При завершении проекта не фиксируются данные о трудозатратах, расписании и качестве.

2) Повторяемый. В некоторой степени отслеживается процесс. Делаются записи о трудозатратах и планах. Функциональность каждого проекта описана в письменной форме. В середине 99 лишь 20% организаций имели 2-й уровень или выше.

3) Установленный. Имеют определенный, документированный и установленный процесс работы, не зависящий от отдельных личностей. Т.е. Вводятся согласованные проф. Стандарты, а разработчики их выполняют. Такие организации

в состоянии достаточно надежно предсказывать затраты на проекты, аналогичные выполненным ранее.

4) Управляемый. Могут точно предсказать сроки и стоимость работ. Есть база данных накопленных измерений. Но нет изменений при появления новых технологий и парадигм.
5) Оптимизированный. Есть постоянно действующая процедура поиска и освоения новых и улучшенных методов и инструментов.

4.5 RUP

В рамках данной лекции внимание акцентируется на возможности поддержки деятельности менеджера проекта в RUP. Для этого рассматривается модель жизненного цикла как основы методических подходов, которые могут развиваться на базе RUP.

RUP (Rational Unified Processing) претендует на роль универсальной основы любых программных разработок по единой («рациональной») схеме. Поэтому естественно, что авторы предлагают общую модель жизненного цикла, которая будет пригодна для всех «рационально» развивающихся проектов. В соответствии с ориентацией преимущественно на стратегию итеративного развития модель должна поддерживать ведение итеративных программных проектов, этапы, выполняемые в ходе итерации, и производственные функции.

Модель задается в виде матрицы интенсивностей функций, выполняемых на этапах (фазах), которые проецируются на итерации (см. Рисунок 4). Для каждой итерации можно указать, в какой фазе она находится в данный момент, а также какая рассматривается функция.

В основе RUP лежат следующие принципы:

- Ранняя идентификация и непрерывное (до окончания проекта) устранение основных рисков.
- Концентрация на выполнении требований заказчиков к исполняемой программе (анализ и построение модели прецедентов (вариантов использования)).
- Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки.
- Компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта.
- Постоянное обеспечение качества на всех этапах разработки проекта (продукта).
- Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам.

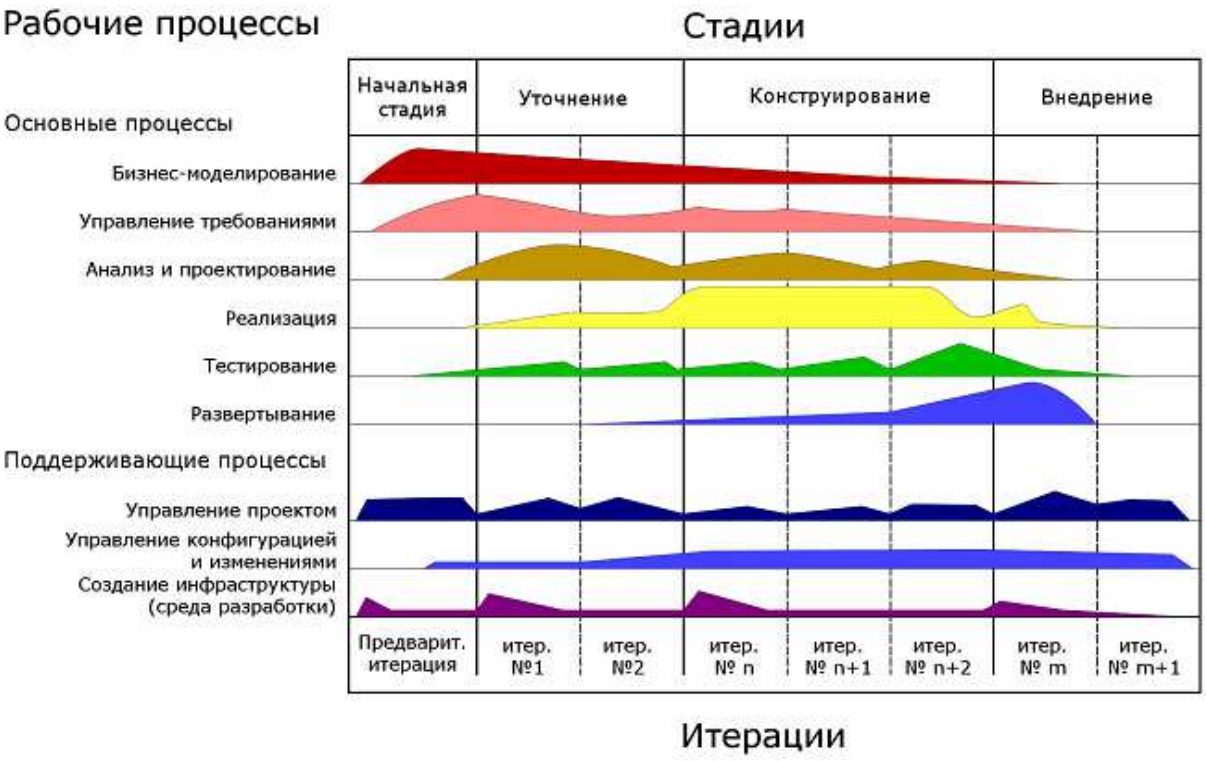


Рисунок 4. Модель жизненного цикла RUP

RUP использует итеративную модель разработки. В конце каждой итерации (в идеале продолжающейся от 2 до 6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта.

Полный жизненный цикл разработки продукта состоит из четырех фаз, каждая из которых включает в себя одну или несколько итераций:

1) Начало (Inception). Здесь:

- Формируются видение и границы проекта.
- Создается экономическое обоснование (business case).
- Определяются основные требования, ограничения и ключевая функциональность продукта.
- Создается базовая версия модели прецедентов.
- Оцениваются риски.

При завершении начальной фазы оценивается достижение *вехи* целей жизненного цикла, которое предполагает соглашение заинтересованных сторон о продолжении проекта.

2) Уточнение (Elaboration). В фазе «Уточнение» производится анализ предметной области и построение исполняемой архитектуры. Это включает в себя:

- Документирование требований (включая детальное описание для большинства прецедентов).
- Спроектированную, реализованную и оттестированную исполняемую архитектуру.

- Обновленное экономическое обоснование и более точные оценки сроков и стоимости.
- Сниженные основные риски.

Успешное выполнение фазы разработки означает достижение *вехи архитектуры жизненного цикла*.

3) Построение (Construction)

В фазе «Построение» происходит реализация большей части функциональности продукта. Фаза Построение завершается первым внешним релизом системы и вехой начальной функциональной готовности (Initial Operational Capability).

4) Внедрение (Transition)

В фазе «Внедрение» создается финальная версия продукта и передается от разработчика к заказчику. Это включает в себя программу бета-тестирования, обучение пользователей, а также определение качества продукта. В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза Внедрение повторяется снова. Выполнение всех целей означает достижение вехи готового продукта (Product Release) и завершение полного цикла разработки.

За рамками модели остаются способы, с помощью которых можно было бы переходить к функциям от фаз жизненного цикла. Моделью не конкретизируются виды работ на этапах – это остается в описательной части документа, представляющего процесс разработки. Время также довольно условно: рассмотрение отдельной функции дает представление о горизонтальном развитии событий (переходов от фазы к фазе), но о возможности совместного выполнения некоторых производственных функций речь не идет (быть может, поэтому функции рассматриваются крупным планом, а схемы модели, иллюстрирующие описание RUP, меняются при обсуждении разных аспектов).

Модель выглядит как универсальная схема: с точностью до наименований она отражает то, что включается в любое производство программного обеспечения. Однако она не приспособлена к включению в схему дополнительных этапов и функций, отражающих специфику конкретного процесса или коллектива разработчиков. Это нарушило бы фиксированную связь между жизненным циклом по RUP с моделями уровня проектирования, которым в обсуждаемом подходе уделено особое внимание.

Безусловно, достоинством RUP является предлагаемый инструментарий моделирования различных аспектов реализуемого программного обеспечения, и именно этот инструментарий применяется чаще всего. Он привлекает разработчиков выразительностью, поддержкой согласованного использования систем моделей, связываемых общей системой понятий.

4.6 MSF

Microsoft Solutions Framework (MSF) – хорошо настраиваемый, масштабируемый, полностью интегрируемый набор процессов разработки программного обеспечения, принципов и проверенных практик, предназначенных для того, чтобы предоставить команде разработчиков программного обеспечения именно тот вид управления проектами, который им больше подходит.

Microsoft Solutions Framework представляет собой хорошо сбалансированный и гибкий набор методик организации процесса разработки, который может быть адаптирован под потребности практически любого коллектива разработчиков и проекта, вне зависимости от его размера и сложности. MSF поддерживает самые различные подходы к организации процесса

разработки, что позволяет команде разработчиков выбирать самый подходящий для них путь. Философия MSF утверждает то, что не существует единой методологии разработки, которая оптимально будет соответствовать требованиям любых проектов. Но, тем не менее, любому проекту необходимо управление. MSF направлена на помощь в обеспечении этого управления. При этом MSF не налагает предписаний, а позволяет команде разработчиков настраивать предоставленные средства. Средства MSF могут быть применены по отдельности или все вместе. Главное – они позволят добиться успеха для многих типов проектов.

Главными принципами MSF являются:

- Производительность: один из ключевых принципов MSF направлен на то, чтобы сделать команду разработчиков более производительной. Производительность в MSF поддерживается хорошо налаженным управлением процесса разработки.
- Интегрируемость: решения и управление представлены инструментальными средствами, посредством плавной интеграции любых наборов инструментальных средств, справки и содержания MSF. Все эти элементы легко обновляются через MSDN.
- Расширяемость: процесс управления и справка полностью настраиваемы в пределах MSF. Разработчики могут выбрать быстрый или более структурированный подход, каждый из которых включает в себя наборы предложенных сценариев, или определить свой собственный подход, используя эти сценарии.

MSF содержит не только рекомендации общего характера, но и предлагает адаптируемую модель коллектива разработчиков, определяющую взаимоотношения внутри коллектива, гибкую модель проектного планирования, основанного на управлении проектными группами, а также набор методик для оценки рисков.

MSF состоит из двух моделей и трех дисциплин. Они подробно описаны в 5 whitepapers:

- модели:
  - модель проектной группы ;
  - модель процессов ;
- дисциплины:
  - дисциплина управление проектами;
  - дисциплина управление рисками;
  - дисциплина управление подготовкой.

Модель процессов MSF (MSF process model) представляет общую методологию разработки и внедрения IT решений. Особенность этой модели состоит в том, что благодаря своей гибкости и отсутствию жестко навязываемых процедур она может быть применена при разработке весьма широкого круга IT проектов. Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной (waterfall) и спиральной (spiral) (Рисунок 5).

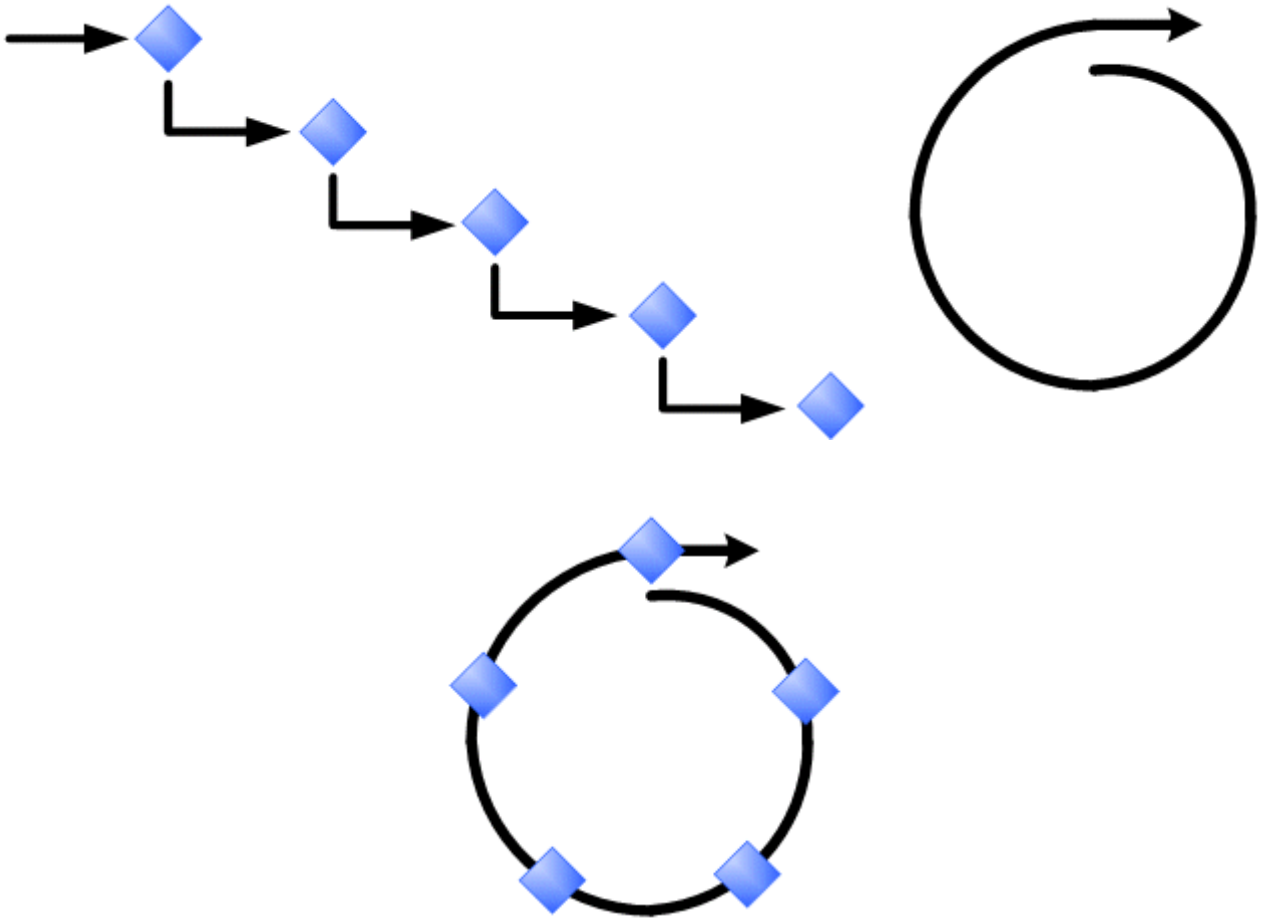


Рисунок 5. Модели процесса: каскадная, спиральная, MSF

Основные принципы модели процессов:

- MSF настаивает на непрерывном взаимодействии с заказчиком в ходе всей работы над проектом.
- Модель процессов MSF считает очень важным открытый обмен информацией как внутри команды, так и с ключевыми заинтересованными лицами.
- Успех коллективной работы над проектом немыслим без наличия у членов проектной группы и заказчика единого видения (shared vision), т.е. четкого, и, самое главное, одинакового, понимания целей и задач проекта.
- MSF настаивает на том, что каждый участник проектной группы должен ощущать ответственность за качество разрабатываемого решения.
- MSF основывается на принципе непрерывной изменяемости условий проекта при неизменной эффективности управленческой деятельности.
- Каждая итерация, каждая фаза процесса создания решения должна заканчиваться некоторым зримым результатом, некоторой вехой (milestone).
- Работа проектной группы в идеале должна быть построена так, чтобы при возникновении такой потребности у заказчика текущее состояние разрабатываемого решения могло быть немедленно внедрено (с той функциональностью, которая в данный момент реализована).

Процесс MSF ориентирован на «вехи» (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо

конечного) результата. Этот результат может быть оценен и проанализирован, что подразумевает ответы на вопросы: «Пришла ли проектная группа к однозначному пониманию целей и рамок проекта?», «В достаточной ли степени готов план действий?», «Соответствует ли продукт утвержденной спецификации?», «Удовлетворяет ли решение нужды заказчика» и т. д.

Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду.

Модель MSF покрывает процесс создания решения с самого его начала и до момента окончательного внедрения. Весь процесс создания решения разбит на пять фаз (Рисунок 6). Каждая из них заканчивается главной вехой, результаты которой становятся видимыми за пределами проектной команды.

Рисунок 6. Фазы и вехи модели процессов MSF

Модель процессов включает такие основные фазы процесса разработки:

- Выработка концепции (Envisioning);
- Планирование (Planning);
- Разработка (Developing);
- Стабилизация (Stabilizing);
- Внедрение (Deploying).

Кроме этого существует большое количество **промежуточных** вех, которые показывают достижение в ходе проекта определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки. Для каждой фазы модели процессов MSF определяет:

- что (какие артефакты) является результатом этой фазы
- над чем работает каждый из ролевых кластеров на этой фазе

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать

разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. С созданием новых версий эволюционирует функциональность решения.

Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. «Живые» документы (living documents) должны изменяться по мере эволюции проекта вместе с изменениями требований к конечному продукту. В рамках MSF предлагается ряд шаблонов стандартных документов, которые являются артефактами каждой стадии разработки продукта и могут быть использованы для планирования и контроля процесса разработки.

К сожалению, авторы предложения MSF не совсем точны в своих оценках модели. Как и в случае с RUP, в этой модели очевидно стремление к универсальности, которое неизбежно приводит к огрублению ситуации в конкретных случаях и к необходимости словесного дополнения схемы. Недостатки моделей, основанных на раскручивающейся спирали, присущи ей в полной мере: невозможность отслеживания временных соотношений между сроками выполнения работ, трудности дополнения специфичных этапов. К тому же ориентация на всеобщность лишает модель и тех преимуществ, которые демонстрирует, например, модель Боэма, снабженная конкретным механизмом интерпретации.

Если обратиться к процессам, которые отражают модель MSF, т.е. к их отдельному описанию, то становится заметным стремление авторов сделать методологию гибкой. К примеру, вот что они пишут о сотрудничестве с заказчиком: «Вовлечение заказчика в проект является необходимым условием успешности проектов. Модель процессов MSF предоставляет заказчику широкий спектр возможностей для уточнения и модификации проектных требований и установки контрольных точек (вех) для мониторинга работы над проектом. В свою очередь, это требует затрат времени со стороны заказчика и взятия им на себя определенных обязательств». Однако далее говорится о том, что «MSF признает первостепенную важность договорных и юридических отношений между заказчиком, его поставщиками и проектной командой и необходимость управления этими отношениями». Только из схемы ни первое, ни второе утверждение извлечь нельзя. И это пример словесного дополнения, к которому приходится прибегать при неудовлетворительном схематическом представлении модели.

Изучение методологии, провозглашаемой MSF, позволяет сделать вывод о том, что ее авторы достаточно тщательно проработали процессы менеджмента, когда основой организации коллектива разработчиков считается проектная группа с рассредоточенными ролями. До уровня стратегии быстрого развития предложения MSF не доходят и занимают промежуточное положение между жесткими и гибкими методологиями.

Лекция 5. Оценка трудоемкости и сроков разработки ПО

Для организации работ над проектом очень важно уметь оценить его трудоемкость. Задачи, решаемые при оценке трудоемкости:

1) разработка бюджета проекта;
2) оценка ресурсов и степени риска (масштаб проекта, количество и квалификация разработчиков, возможность использования готовых решений, качество инструментальных средств и др.);
3) планирование проекта (распределение расходов финансов и других ресурсов по компонентам и этапам проекта);
4) анализ затрат на улучшение качества ПО (оценка целесообразности совершенствования средств и методов разработки).

При определении трудоемкости проекта одинаково недопустимы как недооценка, так и переоценка стоимости, времени и других необходимых ресурсов. Недооценка влечет за собой недостаточную численность проектной команды, нарушение договорных сроков, низкое качество продукта. С другой стороны, переоценка ведет к удорожанию проекта, т.к. при выделении ресурсов сверх необходимого они все равно будут потрачены.

Основные методы оценки трудоемкости:

1) Статистическое моделирование. Метод основан на статистическом анализе данных о ранее выполненных проектах, при этом определяется зависимость трудоемкости проекта от какого-нибудь количественного показателя (например, размера программного кода). Далее этот показатель оценивается для данного проекта, после чего прогнозируется трудоемкость.

2) Экспертные оценки. Проводится опрос нескольких экспертов, знающих область применения разрабатываемого ПО. Каждый из них дает свою оценку трудоемкости. Потом все оценки сравниваются и обсуждаются до тех пор, пока не будет выработана единая оценка трудоемкости.

3) Оценка по аналогии. Метод основан на сравнении планируемого проекта с предыдущими проектами, имеющими сходные характеристики. Могут вырабатываться три оценки: оптимистическая, пессимистическая и средняя.

Существуют и другие методы. В целом большинство моделей оценки трудоемкости использует пять основных параметров:

1) размер конечного продукта (для вновь создаваемых компонентов);
2) особенности организации процесса разработки (наличие формализованных процедур, документирование, разделение функций и др.);
3) возможности персонала, участвующего в разработке ПО (квалификация и стабильность коллектива и пр.);
4) среда разработки проекта, которая включает инструментальные средства и адекватные методики ведения проекта;
5) требуемое качество продукта, особенно в плане надежности, производительности и адаптируемости.

5.1 Метод функциональных точек

Определение числа функциональных точек является методом оценки размера ПО, не зависящим от технологии и языка программирования. В основу метода положена оценка

функциональности разрабатываемой системы. Функциональность определяется на основе выявления и подсчета функциональных типов – групп взаимосвязанных данных, используемых системой, а также элементарных процессов, связанных с вводом и выводом данных.

К функциональным типам относятся (Рисунок 1):

1) Внутренний логический файл (ILF, Internal Logical File) – именованная совокупность данных, поддерживаемая изнутри приложения.

2) Внешний интерфейсный файл (EIF, External Interface File) – именованная совокупность данных, передаваемых другому приложению или получаемых из него.

3) Входной элемент приложения (EI, External Input) – элементарный процесс, связанный с обработкой входной информации – документа или экранной формы.

4) Выходной элемент приложения (EO, External Output) – элементарный процесс, связанный с обработкой выходной информации – отчета, документа или экранной формы.

5) Внешний запрос (EQ, External Query) – элементарный процесс, состоящий из комбинации запрос/ответ, не связанный с вычислением или обновлением ILF.

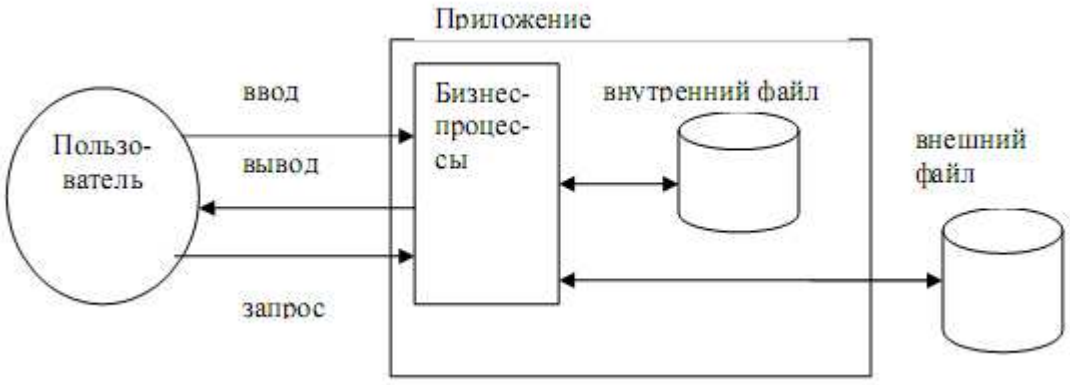


Рисунок 1. Функциональные типы

Для каждого функционального типа определяется сложность (например, по количеству атрибутов объектов), затем подсчитывается количество функциональных точек по всем типам и вводится поправка на сложность системы.

На реализацию проекта размером в одну функциональную точку требуется один день, и они всегда заканчиваются успешно. Таковы небольшие утилиты для временных нужд.

Объем в 10 ФТ – это типичный объем небольших приложений и дополнений, вносимых в готовые системы. Здесь требуется до одного месяца работ, которые также заканчиваются успешно.

Объем в 100 ФТ близок к пределу возможностей программиста-одиночки. Проект доводится до конца в срок до 6 месяцев при успешности в 85 % случаев.

Объем проекта в 1000 ФТ характерен для большинства сегодняшних коммерческих настольных и клиент-серверных приложений. Заметную долю общего объема работ начинает занимать документирование. Для реализации проекта нужна группа примерно из 10 человек, включая программистов, постановщиков, специалистов по тестированию, и около одного года работы. Проваливается 15 % проектов при работе группы и 65 % попыток программистов-одиночек. В 20 % случаев не удается уложиться в сроки, в 25 % проектов отмечается перерасход средств.

Для проекта в 10 000 ФТ требуется около 100 разработчиков. Работы длятся от 1,5 до 5 лет, при этом запланированные сроки чаще всего не выдерживаются Важнейшую роль начинает играть технология тестирования. До 50 % проектов заканчиваются неудачей.

Объем в 100 000 ФТ на сегодня близок к пределу возможностей создания информационных систем. Это объем современных ОС, таких как Microsoft Windows, и крупнейших военных систем. На их создание уходит 5–8 лет. Над проектом трудятся сотни разработчиков, иногда из разных стран. В законченных системах остается много ошибок. До 65 % проектов завершаются неудачно. Во всех успешных проектах используются системы управления конфигурацией. Управление конфигурацией ПО предполагает применение административных и технических процедур на всем ЖЦ ПО для определения состояния компонентов ПО, управления модификациями (версиями) ПО, обеспечения полноты, совместимости и корректности компонентов ПО.

5.2 Методика COCOMO II

Методика COCOMO позволяет оценить трудоемкость и время разработки программного продукта. В модели используется формула регрессии с параметрами, определяемыми на основе отраслевых данных и характеристик конкретного проекта.

Различаются две стадии оценки проекта: *предварительная* оценка на начальной фазе и *детальная* оценка после проработки архитектуры.

Формула оценки трудоемкости проекта в чел.*мес. имеет вид:

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n} EM_i$$

A = 2,94

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

B = 0,91

где

- $SIZE$ – размер продукта в KSLOC
- EM_i – множители трудоемкости
- SF_j – факторы масштаба
- n=7 – для предварительной оценки
- n=17 – для детальной оценки

Главной особенностью методики является то, что для того, чтобы оценить трудоемкость, необходимо знать размер программного продукта в тысячах строках исходного кода (KSLOC, Kilo Source Lines Of Code). Размер программного продукта может быть, например, оценен экспертами с применением метода PERT.

Рассмотрим только случай предварительной оценки трудоемкости программного проекта. Для этой оценки необходимо оценить для проекта уровень семи множителей трудоемкости EM_i:

1) PERS – квалификация персонала (Extra Low – аналитики и программисты имеют низшую квалификацию, текучесть больше 45%; Extra High – аналитики и программисты имеют высшую квалификацию, текучесть меньше 4%)
2) RCPX – сложность и надежность продукта (Extra Low – продукт простой, специальных требований по надежности нет, БД маленькая, документация не требуется; Extra High — продукт очень сложный, требования по надежности жесткие, БД сверхбольшая, документация требуется в полном объеме)
3) RUSE – разработка для повторного использования (Low – не требуется; Extra High – требуется переиспользование в других продуктах)
4) PDIF – сложность платформы разработки (Extra Low – специальные ограничения по памяти и быстродействию отсутствуют, платформа стабильна; Extra High – жесткие ограничения по памяти и быстродействию, платформа нестабильна)
5) PREX – опыт персонала (Extra Low – новое приложение, инструменты и платформа; Extra High – приложение, инструменты и платформа хорошо известны)
6) FCIL – оборудование (Extra Low – инструменты простейшие, коммуникации затруднены; Extra High – интегрированные средства поддержки жизненного цикла, интерактивные мультимедиа коммуникации)
7) SCED – сжатие расписания (Very Low – 75% от номинальной длительности; Very High – 160% от номинальной длительности)

Влияние множителей трудоемкости в зависимости от их уровня определяется их числовыми значениями, которые представлены в матрице, приведенной ниже (Таблица 1).

Таблица 1. Значения множителей трудоемкости, в зависимости от оценки их уровня

	Оценка уровня множителя трудоемкости						
	Extra Low	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.5
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE	n/a	n/a	0.95	1.00	1.07	1.15	1.24
PDIF	n/a	n/a	0.87	1.00	1.29	1.81	2.61
PREX	1.59	1.33	1.22	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	n/a	1.43	1.14	1.00	1.00	1.00	n/a

Из этой таблицы, в частности, следует, если в нашем проекте низкая квалификация аналитиков, то его трудоемкость возрастет примерно в 4 раза по сравнению с проектом, в котором участвуют аналитики экстра-класса.

В методике используются пять факторов масштаба SF_j, которые определяются следующими характеристиками проекта:

1) PREC – прецедентность, наличие опыт аналогичных разработок (Very Low – опыт в продукте и платформе отсутствует; Extra High – продукт и платформа полностью знакомы)
2) FLEX – гибкость процесса разработки (Very Low – процесс строго детерминирован; Extra High – определены только общие цели).

3) RESL – архитектура и разрешение рисков (Very Low – риски неизвестны/не проанализированы; Extra High – риски разрешены на 100%)
4) TEAM – сработанность команды (Very Low – формальные взаимодействия; Extra High — полное доверие, взаимозаменяемость и взаимопомощь).
5) PMAT – зрелость процессов (Very Low – CMM Level 1; Extra High – CMM Level 5)

Значение фактора масштаб, в зависимости от оценки его уровня, приведены в Таблица 2

Таблица 2. Значение фактора масштаба, в зависимости от оценки его уровня

Фактор масштаба	**Оценка уровня фактора**					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Методика COCOMO II определяет следующую последовательность вычисления трудоемкости проекта при многокомпонентной разработке.

1) Суммарный размер продукта рассчитывается, как сумма размеров его компонентов:

$$SIZE^A = \sum_{k=1}^{N} SIZE_k$$

2) Базовая трудоемкость проекта рассчитывается по формуле:

$$PM^B = A \times \left(SIZE^A\right)^E \times SCED$$

3) Затем рассчитывается базовая трудоемкость каждого компонента:

$$PM_k^B = PM^B \times \frac{SIZE_k}{SIZE^A}$$

4) На следующем шаге рассчитывается оценка трудоемкости компонентов с учетом всех множителей трудоемкости, кроме множителя *SCED.*

$$PM_k^{'} = PM_k^B \times \prod_{i=1}^{6} EM_i$$

5) И, наконец, итоговая трудоемкость проекта определятся по формуле:

$$PM = \sum_{k=1}^{N} PM_k^{'}$$

Длительность проекта в методике COCOMO II рассчитывается по формуле:

$$TDEV = C \times \left(PM_{NS}\right)^{D+0,2\times 0.01\times \sum_{j=1}^{5} SF_j} \times \frac{SCED}{100}$$

где

- *C = 3,67; D = 0,28;*
- PM_{NS} *– трудоемкость* проекта без учета множителя *SCED*, определяющего сжатие расписания.

5.3 Методология RAD – Rapid Application Development

Rapid Application Development (RAD, быстрая разработка приложений) – это жизненный цикл процесса проектирования, созданный для достижения более высоких скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию.

RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки. Технология RAD предусматривает активное привлечение заказчика уже на ранних стадиях – обследование организации, выработка требований к системе. Причины популярности RAD вытекают из тех преимуществ, которые обеспечивает эта технология.

Наиболее существенными из них являются:

- высокая скорость разработки;
- низкая стоимость;
- высокое качество.

Последнее из указанных свойств подразумевает полное выполнение требований заказчика как функциональных, так и нефункциональных, с учетом их возможных изменений в период разработки системы, а также получение качественной документации, обеспечивающей удобство эксплуатации и сопровождения системы. Это означает, что дополнительные затраты на сопровождение сразу после поставки будут значительно меньше. Таким образом, полное время от начала разработки до получения приемлемого продукта при использовании этого метода значительно сокращается.

Применение технологии RAD целесообразно, когда:

- требуется выполнение проекта в сжатые сроки (90 дней). Быстрое выполнение проекта позволяет создать систему, отвечающую требованиям сегодняшнего дня. Если система проектируется долго, то весьма высока вероятность, что за это время существенно изменятся фундаментальные положения, регламентирующие деятельность организации, то есть, система морально устареет еще до завершения ее проектирования.
- нечетко определены требования к ПО. В большинстве случаев заказчик весьма приблизительно представляет себе работу будущего программного продукта и не может четко сформулировать все требования к ПО. Требования могут быть вообще не определены к началу проекта либо могут изменяться по ходу его выполнения.
- проект выполняется в условиях ограниченности бюджета. Разработка ведется небольшими RAD группами в короткие сроки, что обеспечивает минимум трудозатрат и позволяет вписаться в бюджетные ограничения.
- интерфейс пользователя (GUI) есть главный фактор. Нет смысла заставлять

пользователя рисовать картинки. RAD технология дает возможность продемонстрировать интерфейс в прототипе, причем достаточно скоро после начала проекта.

- проект большой, но поддается разделению на более мелкие функциональные компоненты. Если предполагаемая система велика, необходимо, чтобы ее можно было разбить на мелкие части, каждая из которых обладает четкой функциональностью. Они могут выпускаться последовательно или параллельно (в последнем случае привлекается несколько RAD групп).
- ПО не обладает большой вычислительной сложностью.

RAD-технология не является универсальной, то есть ее применение целесообразно не всегда. Например, в проектах, где требования к программному продукту четко определены и не должны меняться, вовлечение заказчика в процесс разработки не требуется и более эффективной может быть иерархическая разработка (каскадный метод). То же касается проектов, ПО, сложность которых определяется необходимостью реализации сложных алгоритмов, а роль и объем пользовательского интерфейса невелик.

Принципы RAD технологии направлены на обеспечение трех основных ее преимуществ – высокой скорости разработки, низкой стоимости и высокого качества. Достигнуть высокого качества программного продукта весьма непросто и одна из главных причин возникающих трудностей заключается в том, что разработчик и заказчик видят предмет разработки (ПО) по-разному.

Главная идея RAD технологии состоит в том, чтобы как можно быстрее донести до заказчика результаты разработки, пусть и не в полном виде. Например, реализация только пользовательского интерфейса и предъявление его заказчику позволяет уже на ранней стадии разработки получить замечания по экранным и отчетным формам и внести необходимые коррективы. В этом случае значительно возрастает вероятность успеха проекта, то есть возникает уверенность в том, что конечный продукт будет делать именно то, что ожидает заказчик. Кроме того, не следует забывать и тот факт, что разница стоимости ошибки определения требований в начале проекта и в конце равна 1:200.

Основные принципы RAD можно сформулировать следующим образом:

- Работа ведется группами. Типичный состав группы – руководитель, аналитик, два программиста, технический писатель. Если проект сложный, то для него может быть выделено несколько RAD-групп. Разработка проекта выполняется в условиях тесного взаимодействия между разработчиками и Заказчиком.
- Разработка базируется на моделях. Моделирование позволяет оценить проект и выполнить его декомпозицию на составные части, каждая из которых может разрабатываться отдельной RAD-группой.
- Итерационное прототипирование. Разработка системы и предъявление ее заказчику осуществляется в виде последовательности развиваемых прототипов. Любой из прототипов реализует определенную часть функциональности, требуемой от конечного продукта. При этом каждый последующий прототип включает всю функциональность, реализованную в предыдущем прототипе, с добавлением новой. Число прототипов определяется на основе учета разных параметров – размера проекта, анализа рисков, пожеланий заказчика и т. д. Традиционно для проектов ПО средней сложности разрабатываются три прототипа. Первый содержит весь пользовательский интерфейс с нулевой функциональностью. Он дает возможность собрать замечания заказчика и после их устранения утвердить у него экранные и отчетные формы. Второй прототип содержит реализованную на 70-80% функциональность системы, третий – полностью реализованную функциональность.

Основаниями для очередной итерации являются:

1) *Замечания заказчика*. Привлечение заказчика и конечного пользователя к оценке выходных результатов прототипа с эффективной обратной связью с командой разработчиков является гарантией того, что созданная система будет делать то, что требуется заказчику. Если замечания носят характер исправлений, они учитываются в следующем прототипе, если же изменяются требования, то выполняется переоценка проекта и корректируются сроки и стоимость проекта.
2) *Детализация*. Выполняется программирование нереализованной части системы в соответствии с составленным планом.
3) *Анализ результатов программирования*. Исправляются ошибки, повышается эффективность программного кода и т. д.

- RAD группа всегда работает только над одним прототипом. Это обеспечивает единство целей, лучшую наблюдаемость и управляемость процессом разработки, что в итоге повышает качество конечного продукта. Соответственно используемые инструментальные средства должны обеспечивать групповую разработку и конфигурационное управление проектом.
- Если проект сложный, то для него может быть выделено несколько RAD групп. Большие системы разбиваются на подсистемы. Каждая подсистема разрабатывается независимой группой. Ключ успеха – правильное разбиение системы на подсистемы. Команды должны использовать общие стандарты. Обязательно финальное тестирование полной системы.
- Обязательное использование инструментальных средств, автоматизирующих процесс разработки, и методик их использования, следствием чего является сокращение сроков разработки и повышение качества конечного продукта.

Принципы RAD применяются не только при реализации, но и распространяются на все этапы жизненного цикла, в частности на этап обследования организации, построения требований, анализ и дизайн.

Технология RAD обеспечивает:

- быстроту продвижения программного продукта на рынок;
- интерфейс, устраивающий пользователя;
- легкую адаптируемость проекта к изменяющимся требованиям;
- простоту развития функциональности системы.

Жизненный цикл создания ИС на основе RAD-технологии предполагает после формирования технического задания и декомпозиции системы независимую разработку подсистем с последующей сборкой, тестированием и внедрением комплексной ИС (Рисунок 2).

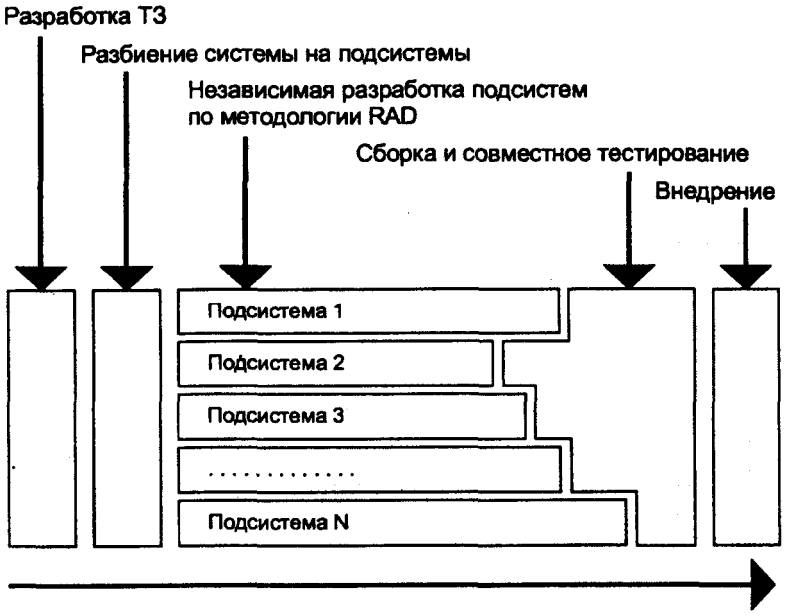


Рисунок 2. Жизненный цикл создания ИС на основе RAD-технологии

Накопленный опыт использования RAD-технологии показывает, что существуют два базовых варианта организации технологического процесса проектирования с использованием систем-прототипов.

В первом варианте создание системы-прототипа используется для лучшей спецификации требований к разработке ИС, после разработки которых сам прототип оказывается ненужным. В этом случае традиционно разрабатывается «Постановка задачи», документация которой является спецификацией системы-прототипа. После демонстрации пользователю и доработки прототипа разрабатывается новая «Постановка задачи», которая служит основой создания действующей ИС.

Основным недостатком первого варианта использования прототипирования является неэффективное использование системы-прототипа, а именно: прототипы не используются в дальнейшей разработке ИС после того, как выполнили свою первую задачу - устранили неясности в проекте.

Второй вариант предполагает итерационное развитие системы-прототипа в готовый для эксплуатации программный продукт. Итерации разработки системы-прототипа включают создание/модификацию системы-прототипа, ее демонстрацию пользователю и согласование, разработку новых спецификаций-требований к системе, новую модификацию и т.д., пока не будет создано готовое приложение. Документацию компонентов системы-прототипа непосредственно составляют спецификации, которые являются требованиями к программной реализации системы и определяют характер взаимоотношений с заказчиком на этапе сдачи готовой системы.

Итерационное использование прототипного подхода к разработке ИС обеспечивает экономию ресурсов на проектирование, а самое главное, - резкое сокращение времени на разработку и внедрение готовой к эксплуатации системы. При этом основным достоинством прототипной технологии является значительное снижение объема доработок ИС при ее внедрении, который для традиционных методов проектирования, как показывает опыт, соразмерен с затратами на первоначальную реализацию.

Лекция 6. Гибкое управление разработкой

6.1 Методологии быстрого развития проектов

Как утверждают сторонники быстрого развития, их методологии не нуждаются в том, чтобы четко фиксировать этапы развития разработки программного проекта. Однако они понимают, что само понятие жизненного цикла полезно для представления процесса разработки в концептуальном плане. Что же касается деятельности менеджера, то в этом подходе в противовес жестким методологиям провозглашаются самодисциплина и сотрудничество вместо дисциплины и подчинения; планирование, контрольные и другие функции носят здесь такой характер, который позволяет менеджеру в большей мере сосредоточиться на руководстве командой, чем на управлении. В результате отслеживание процесса не требует, к примеру, специальных документов о достигнутых результатах и проблемах, для которых нужна специальная поддержка. По этой причине модели жизненного цикла быстрого развития не претендуют на инструментальность, и в таком ключе их рассматривать не имеет смысла. Тем не менее понятия контрольных точек и контрольных мероприятий, распределения ресурсов, оценки остаются, хотя их содержание становится менее формализованным.

Жизненный цикл любой методологии быстрого развития можно описать следующим образом.

- **Начальная фаза**. Она выделена, поскольку приходится выполнять работы, которые не являются характерными для основного процесса.
- **Серия** максимально коротких **итераций**, состоящих из шагов:
  - выбор реализуемых требований (в экстремальном программировании – пользовательских историй);
  - реализация только отобранных требований;
  - передача результата для практического использования;
  - короткий период оценки достигнутого (в зависимости от объема работ периода его можно назвать этапом или контрольным мероприятием).
- **Фаза заключительной оценки** разработки проекта.

Реальные быстрые методологии конкретизируют эту схему, дополняют ее теми или иными методиками.

6.2 Экстремальное программирование

Экстремальное программирование (extreme programming, XP) – методология гибкой разработки программного обеспечения, ориентированная на быстрый выпуск новых версий в условиях часто изменяющихся требований, с обязательным включением заказчика в процесс разработки.

В основе экстремального программирования лежит несколько принципов, определяющих, что, когда и как должно делаться.

1) Экстремальный цикл

В основе экстремального программирования – очень короткий, постоянно повторяющийся цикл разработки, составляющий одну-три недели. К концу каждого цикла необходимо иметь полностью рабочий, функциональный и протестированный релиз приложения. Эти циклы должны быть повторяющимися и бесперебойными на протяжении всего проекта.

Предпосылкой для такого режима работы является многократно проверенный факт, что требования редко бывают полными, своевременными и корректными. Иными словами, как бы хорошо вы ни планировали свое приложение, имейте в виду, что его 100% придется переделывать. Более того, его, возможно, придется переделывать даже на завершающей стадии. Не откладывайте переделки на конец работы, делайте их регулярно.

Как следствие постоянно изменяющихся требований следует другой принцип – позднее принятие решений.

2) Позднее принятие решений

«Поздний анализ» означает «принимайте конкретные решения только тогда, когда это нужно». В большинстве случаев принятые на начальной стадии решения относительно кода приходилось отменять под влиянием новых требований или других обстоятельств. Не принимайте никаких решений по поводу кода, которого у вас еще нет, – тогда вы развяжете себе руки при реализации текущих задач. Как правило, многое из запланированного оказывается вообще не востребованным. Поэтому планировать кодирование полезно перед началом каждого цикла, но не «раз и навсегда».

С другой стороны, любая начатая часть работы, любая подсистема должна быть закончена прежде, чем начнется работа над другими секциями кода. Такая методика известна как кодирование в глубину.

3) Кодирование в глубину

Кодирование в глубину (по аналогии с названием метода обхода бинарного дерева) обозначает, что в течение цикла должна быть полностью разработана и протестирована отдельная функциональность и проигнорированы соседние с ней области. Подразумевается, что готовая часть будет включать прикладную логику, пользовательский интерфейс, документацию и набор тестовых заданий для демонстрации работоспособности.

Из готовых, законченных функциональностей складываются истории пользователя, и именно «закрытие» отдельного класса проблем пользователя в самый быстрый срок должно стать основной целью. В противном случае может сложиться ситуация, когда на 90% готовое приложение нельзя показать заказчикам и пользователям, поскольку оно до конца не выполняет ни одной функции.

Помимо концентрации на одной задаче в каждом конкретном цикле важное значение имеют также скорость проекта и фактор загрузки.

4) Идеальный день разработчика и фактор загрузки

Важным первичным инструментом при расчетах является идеальный день разработчика – то есть день, когда разработчик полностью концентрируется на решении проблем проекта. Это тот срок, в который разработчик может выполнить заданный объем работы при максимальной загрузке. Но было бы большой ошибкой использовать разработчика в этом режиме без крайней необходимости. Ритм работы экстремального программирования отнюдь не экстремален – и даже не должен подходить к отметке «горячо», по крайней мере в начале разработки.

Фактор загрузки – это отношение реальных календарных рабочих дней к идеальным дням разработчика. Нормальными считаются факторы от двух до пяти, причем чем больший фактор вы можете позволить – тем больший запас адреналина есть в запасе у вашей команды. Опытные менеджеры используют «три» как стартовое значение, но для новых неосвоенных технологий следует использовать четыре или пять.

Другим временным фактором является скорость проекта.

5) Скорость проекта

Скорость проекта – это скорость реализации частей программы, определенных для заданного цикла. В качестве основных ориентиров прогресса в разработке выступают реализации историй пользователей.

6) История пользователей

История пользователя – это компактный документ (предположительно около трех предложений) составленный пользователем и описывающий одну отдельную операцию для данного пользователя в духе «я захожу в программу и...». В отличие от глобальных Use Case, где рассматриваются целые классы пользователей, историю пользователя легко определить, спланировать на конкретный цикл и реализовать в определенный срок. Скажем, «ввод накладной» значительно более ясен и детерминирован, чем «обработка входных документов».

Истории пользователей до последнего момента не принимают детального вида, в духе позднего принятия решений. Реализация истории должна быть ограничена по срокам от одной до трех недель разработки – в противном случае такая история должна быть разбита на более мелкие. Главное – избежать ситуации, когда нечто достаточно долго делается без обратной связи, поскольку все сделанное потенциально является предметом критики и переработки.

Считается, что 80 ± 20 историй пользователей являются идеальным числом для составления плана нового релиза.

7) План релиза

План релиза, утверждаемый на специальном совещании, дает точный ответ на вопрос, какие именно истории пользователей будут реализованы в данном релизе. Преимущество отдается небольшим инкрементальным релизам. Выбранные к реализации истории транслируются в конкретные задания программирования, такие как создание формы ввода или процедуры запроса к БД. Обычно после нескольких итераций оценки необходимых операций осуществляются очень точно. Как только план выполнения итераций выходит из-под контроля и, по крайней мере, после каждых нескольких удачных итераций повторно собираются совещания по поводу нового плана релиза.

Выбранные истории являются основой для планов итераций.

8) План итераций

План итераций ограничивает количество заданий, которые будут выполняться в данной итерации. Выборка производится на основании текущей скорости проекта, то есть на основе оценки идеального срока, умноженного на фактор загрузки. Истории пользователей получают приоритеты внутри цикла и трансформируются в задачи разработки, каждая из которых выполняется в течение одного-трех идеальных рабочих дней.

Если в результате детализации ожидаемое время разработки превосходит время цикла, то некоторые истории переносятся на более поздний срок. Этот эффект снежного кома – вполне обычная практика, поскольку детальные задачи часто распадаются на отдельные части, когда сумма времени для каждой превосходит время для целого.

Не стоит искать виновных в этой ситуации. Такое свойство планирования, как недооценка деталей, это и есть та причина, по которой не производится предварительное планирование. Детальный предварительный план всегда будет пересмотрен в будущем, и поэтому изначально и гарантированно нереален. Планирование производится только на основании предыдущего цикла, с коррекцией скорости проекта и с учетом перенесенных заданий.

Параллельно с выборкой историй пользователей план предусматривает создание набора тестов приемки, которые будут сопровождать код в процессе всех последующих сборок.

9) Тесты приемки

Тесты приемки создаются на основании историй пользователей и желательно до, а не после создания программных модулей. Без прохождения тестов история не может считаться реализованной ни в какой мере. Фактически, тесты приемки – это те же истории пользователей, но умноженные на возможные ошибки ввода и другие варианты поведения системы, то есть рассматриваются различные варианты конкретной истории. Естественно, что для многократного тестирования необходимо создать автоматические процедуры, вводящие тестовые наборы и ведущие журналы тестов.

Тестирование – не приложение к приложению, а, скорее, наоборот. Часто контроль качества (QA) и составление тестов возлагается на отдельную группу, в состав которой входят представители заказчиков.

10) Представители заказчиков

Представители заказчиков являются важнейшим звеном успешной разработки по технологии XP. Представители должны не просто контактировать, но буквально физически присутствовать в непосредственной близости и работать в команде разработчиков. Любая проблема должна быть обнаружена на самом раннем этапе, любые пожелания или вопросы должны решаться в реальном времени. Представители заказчика являются источником историй пользователей и тестовых наборов данных, они принимают участие в планировании плана релизов. Кроме того, открытый процесс позволяет инспектировать спорные участки кода в любой момент времени, создавая полностью прозрачную атмосферу между разработчиками и заказчиками.

Хотя это не очевидно, но практически происходит экономия времени заказчика, который все время находится в курсе дел – дополнительно время экономится на детальных спецификациях в начале работы, поскольку любой аспект можно выяснить в процессе работы. Впоследствии не придется инструктировать персонал заказчика, поскольку заказчик уже располагает высококачественными специалистами по данному продукту.

При этом многие документы-посредники становятся ненужными, поскольку многое решается устно, без вовлечения технических и бюрократических механизмов. Значение имеют только конечные результаты работы – но не промежуточные решения и дискуссии, так что нет необходимости документировать все возможные ходы мысли и альтернативы.

Помимо тесного взаимодействия с заказчиками, особого внимания требует и структура группы разработчиков.

11) Структура группы разработчиков

Для быстрой разработки применяется особая методология организации работы, основанная на теории психологии малых групп. Основной принцип: все разработчики должны быть доступны друг для друга, как организационно, так и физически, – преимущественно желательно располагать группу в одной большой комнате. Структура в группе – одноранговая, то есть существует только профессиональный авторитет, но не административное деление. Доказано, что самая продуктивная обстановка – умеренный шум большой рабочей комнаты, тишина или музыкальное сопровождение дают худшие результаты.

Все производственные вопросы решаются в самой группе, включая используемые методы, инструменты или технологии – эти параметры решения задачи не могут быть навязаны извне: как правило, профессионалы в коллективе решают поставленные задачи оптимально. Кроме того, каждый разработчик самостоятельно выбирает подходящие задания, в зависимости от потребности группы и собственных способностей.

Одной из самых качественных технологий программирования на сегодня является парная технология, когда один программист вводит код, а другой при этом смотрит на экран и по ходу корректирует его или задает вопросы относительно реализации. Такая подстрочная

критика кода, как показывает практика, в несколько раз повышает качество начального кода, сводя возможность ошибок в начальном коде к минимуму.

Второй принцип – принцип замены партнеров – означает, что пары меняют партнеров один-два раза в течение дня, причем группы работают над различными частями системы. Кажется, что такая метода не позволяет сконцентрироваться на отдельной области – однако практика опровергает это. Концентрация на одной области приводит к укоренению стиля, не всегда оптимального, а отсутствие критики кода может привести «хакера» к тяжело обнаружимым ошибкам. Постоянная смена области приложения, напротив, повышает не только квалификацию в результате обмена опытом, но также и мотивацию повышения собственного мастерства.

В обстановке групповой работы важное значение получает максимальная простота и эффективность используемого кода.

12) Простота и эффективность используемого кода

Часто предоставленный сам себе разработчик решает использовать новую технологию или инструмент, который, по его мнению, обещает дополнительную производительность или выгоду. На самом деле это редко оправдывается, а в атмосфере, когда ваш код завтра будет прочитан и использован партнерами по команде, у вас просто не остается возможностей для экспериментов за чужой счет. Фактически, все члены команды изначально должны выбрать технологии, которыми владеют все или большинство членов команды.

К сожалению, программный код, созданный по самым лучшим образцам, со временем имеет тенденцию к деградации под воздействием изменений и исправлений. Поэтому нужно взять за правило производить постоянный и бескорыстный рефракторинг.

13) Рефракторинг

Это наведение порядка в коде, переработка отдельных файлов и их групп с целью наведения порядка, удаления максимального количества ненужных фрагментов, объединения классов на основе схожей функциональности, коррекция комментариев, осмысленное переименование объектов и так далее. Рефракторинг должен стать лучшим отдыхом программиста в промежутке между решением сложных проблем.

Не следует также забывать и о постоянном тестировании модулей.

14) Тестирование модулей

В отличие от тестов приемки, создаваемых заказчиками и отражающих истории пользователей, тестирование модулей – это сугубо технологическая проверка корректности классов на основании готовой или созданной специально для этих целей автоматизированной системы. Настоящая методика тестирования предусматривает создание сначала тестов и только после – самих классов, что исключает «подгонку» тестов к работающим прототипам и уклонение от острых ситуаций.

Классы-тесты должны быть неотъемлемой частью библиотеки или пакета, их отсутствие вызывает сомнения в работоспособности класса. Создаваемый для приложения код должен проходить все более сложные тесты – и таким образом изначально создаваться для некоторого реального окружения. Игнорирование этого принципа может усложнить тестирование впоследствии, когда уже трудно восстановить работу многих модулей.

Тесты служат четырем целям: во-первых, они, собственно, помогают протестировать функциональность; во-вторых, это естественная форма проектной документации, то есть на вопрос «что делает этот класс» можно ответить «пытается пройти набор функциональных тестов». Третье: тесты – это готовая часть описательной документации, сопровождающей класс после разработки, всегда отвечающая на вопрос «как использовать данный класс». И, наконец, это средство групповой разработки.

Выгода от написания модульных тестов хорошо проявляется в отношении группового авторства: фиксированный набор тестов, созданный модератором класса, всегда может гарантировать, что последующие изменения, произведенные другими членами команды, по-прежнему позволяют классу удовлетворять тестам.

15) Групповое авторство

Одной из проблем программирования является недоступность по той или иной причине разработчика определенного фрагмента кода, в результате чего его поддержка становится сложной или невозможной.

В обстановке, когда «все делают все», идеи свободно высказываются и распространяются в группе. Более того, каждый может вносить изменения в любую строку кода. Зависимость группы от одного, даже высококвалифицированного человека – весьма опасный путь. Участие в обсуждении архитектуры всех сотрудников не только повышает самооценку, улучшает квалификацию, но и приводит к значительно лучшим результатам разработки. И поскольку каждый принимает участие в обсуждении архитектурных и организационных вопросов, то вполне логично, что и собственность на полученный продукт будет коллективной.

6.3 Scrum

Scrum – одна из самых популярных методологий гибкой разработки. Одна из причин ее популярности – простота. Схема Scrum приведена на Рисунок 1.

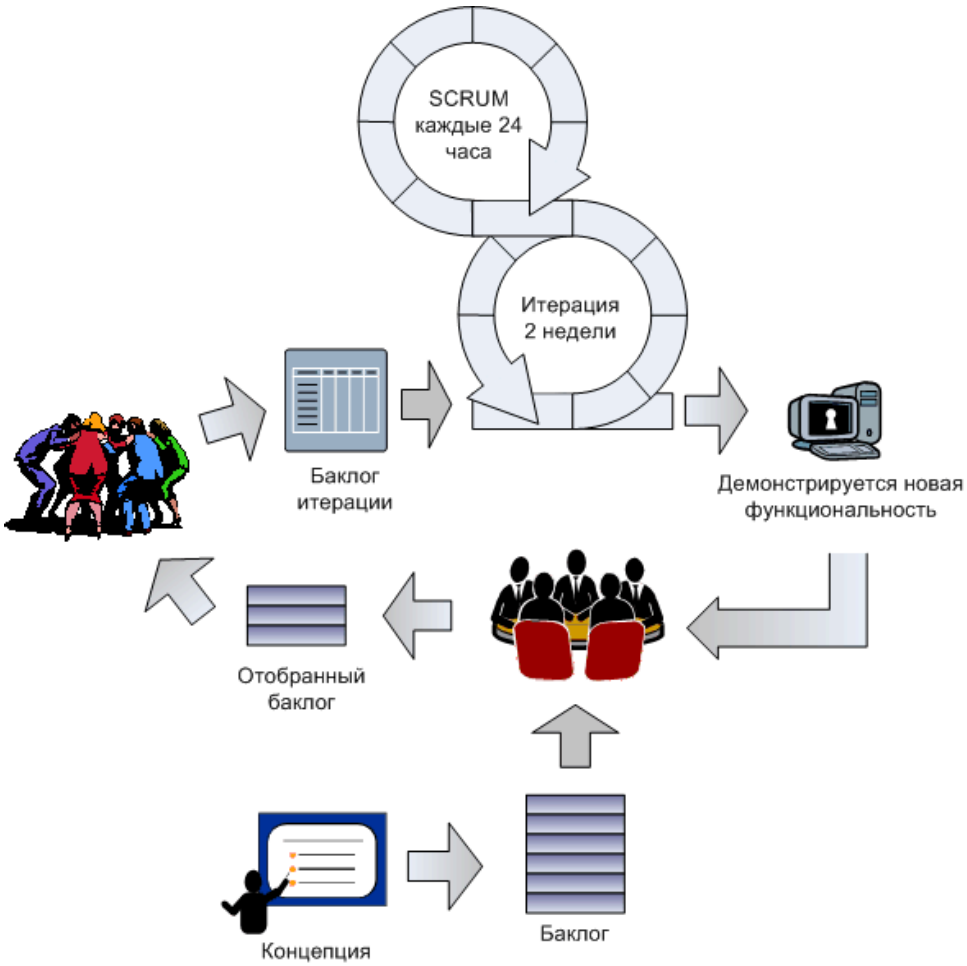


Рисунок 1. Основа Scrum

В методологии Scrum всего три роли.

1) Скрам Мастер (Scrum Master) – самая важная роль в методологии. Скрам Мастер отвечает за успех Scrum в проекте. По сути, Скрам Мастер является интерфейсом между менеджментом и командой. Как правило, эту роль в проекте играет менеджер проекта или тимлид. Важно подчеркнуть, что Скрам Мастер не раздает задачи членам команды. В Agile команда является самоорганизующейся и самоуправлямой.

Основные обязанности Скрам Мастера:

- создает атмосферу доверия;
- участвует в митингах в качестве фасилитатора;
- устраняет препятствия;
- делает проблемы и открытые вопросы видимыми;
- отвечает за соблюдение практик и процесса в команде.

Скрам Мастер ведет Daily Scrum Meeting и отслеживает прогресс команды при помощи Sprint Backlog, отмечая статус всех задач в спринте. ScrumMaster может также помогать Product Owner создавать Backlog для команды

2) Product Owner – это человек, отвечающий за разработку продукта. Как правило, это product manager для продуктовой разработки, менеджер проекта для внутренней разработки и представитель заказчика для заказной разработки. Product Owner - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один человек, а не группа или комитет.

Обязанности Product Owner таковы:

- отвечает за формирование product vision;
- управляет ROI;
- управляет ожиданиями заказчиков и всех заинтересованных лиц;
- координирует и приоритизирует Product backlog;
- предоставляет понятные и тестируемые требования команде;
- взаимодействует с командой и заказчиком;
- отвечает за приемку кода в конце каждой итерации.

Product Owner ставит задачи команде, но он не вправе ставить задачи конкретному члену проектной команды в течение спринта.

3) Команда (Team). В методологии Scrum команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Product Owner. Работа команды оценивается как работа единой группы. В Scrum вклад отдельных членов проектной команды не оценивается, так как это разваливает самоорганизацию команды.

Обязанности команды таковы:

- отвечает за оценку элементов баклога;
- принимает решение по дизайну и имплементации;
- разрабатывает софт и предоставляет его заказчику;

– отслеживает собственный прогресс (вместе со Скрам Мастером);

– отвечает за результат перед Product Owner.

Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные размер команды - 7 ± 2.

Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками – разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды. Команда самоорганизуется для выполнения конкретных задач в проекте, что позволяет ей гибко реагировать на любые возможные задачи.

Для облегчения коммуникаций команда должна находиться в одном месте (colocated). Предпочтительно размещать команду не в кубиках, а в одной общей комнате для того, чтобы уменьшить препятствия для свободного общения. Команде необходимо предоставить все необходимое для комфортной работы, обеспечить досками и флипчартами, предоставить все необходимые инструменты и среду для работы.

В рамках методологии Scrum разрабатываются следующие артефакты: Product Backlog и Sprint Backlog.

Product Backlog – это приоритезированный список имеющихся на данный момент бизнес-требований и технических требований к системе. Product Backlog постоянно пересматривается и дополняется – в него включаются новые требования, удаляются ненужные, пересматриваются приоритеты. За Product Backlog отвечает Product Owner. Он также работает совместно с командой для того, чтобы получить приближенную оценку на выполнение элементов Product Backlog для того, чтобы более точно расставлять приоритеты в соответствии с необходимым временем на выполнение.

Sprint Backlog содержит функциональность, выбранную Product Owner из Product Backlog. Все функции разбиты по задачам, каждая из которых оценивается командой. Каждый день команда оценивает объем работы, который нужно проделать для завершения задач. Сумма оценок оставшейся работы может быть построена как график зависимости от времени. Такой график называется Sprint Burndown chart. Он демонстрирует прогресс команды по ходу спринта.

В Scrum итерация называется Sprint. Ее длительность составляет 1 месяц (30 дней). Результатом Sprint является готовый продукт (build), который можно передавать (deliver) заказчику.

Короткие спринты обеспечивают быстрый feedback проектной команде от заказчика. Заказчик получает возможность гибко управлять scope системы, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в Product Backlog, приоритезируются наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов.

Каждый спринт представляет собой маленький «водопад». В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта. Scope спринта должен быть фиксированным. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в спринте. Это означает, что Sprint Backlog не может быть изменен никем, кроме команды.

В начале каждого спринта проводится **планирование спринта**. В планировании спринта участвуют заказчики, пользователи, менеджмент, Product Owner, Скрам Мастер и команда. Планирование спринта состоит из двух последовательных митингов.

1) Планирование спринта, митинг первый

   Участники: команда, Product Owner, Scrum Master, пользователи, менеджемент

   Цель: Определить цель спринта (Sprint Goal) и Sprint Backlog –функциональность, которая будет разработана в течение следующего спринта для достижения цели спринта.

   Артефакт: Sprint Backlog

2) Планирование спринта, митинг второй

   Участники: Скрам Мастер, команда

   Цель: определить, как именно будет разрабатываться определенная функциональность для того, чтобы достичь цели спринта. Для каждого элемента Sprint Backlog определяется список задач и оценивается их продолжительность.

   Артефакт: в Sprint Backlog появляются задачи

Если в ходе спринта выясняется, что команда не может успеть сделать запланированное на спринт, то Скрам Мастер, Product Owner и команда встречаются и выясняют, как можно сократить scope работ и при этом достичь цели спринта.

Остановка спринта (Sprint Abnormal Termination) производится в исключительных ситуациях. Спринт может быть остановлен до того, как закончатся отведенные 30 дней. Спринт может остановить команда, если понимает, что не может достичь цели спринта в отведенное время. Спринт может остановить Product Owner, если необходимость в достижении цели спринта исчезла.

После остановки спринта проводится митинг с командой, где обсуждаются причины остановки спринта. После этого начинается новый спринт: производится его планирование и стартуются работы.

Daily Scrum Meeting – этот митинг проходит каждое утро в начале дня. Он предназначен для того, чтобы все члены команды знали, кто и чем занимается в проекте. Длительность этого митинга строго ограничена и не должна превышать 15 минут. Цель митинга – поделиться информацией. Он не предназначен для решения проблем в проекте. Все требующие специального обсуждения вопросы должны быть вынесены за пределы митинга. Скрам митинг проводит Скрам Мастер. Он по кругу задает вопросы каждому члену команды

– Что сделано вчера?

– Что будет сделано сегодня?

– С какими проблемами столкнулся?

Демо и ревью спринта (рекомендованная длительность: 4 часа). Команда демонстрирует инкремент продукта, созданный за последний спринт. Product Owner, менеджмент, заказчики, пользователи, в свою очередь, его оценивают. Команда рассказывает о поставленных задачах, о том как они были решены, какие препятствия были у них на пути, какие были приняты решения, какие проблемы остались нерешенными. На основании ревью принимающая сторона может сделать выводы о том, как должна дальше развиваться система. Участники миитинга делают выводы о том, как шел процесс в команде и предлагает решения по его улучшению.

Скрам Мастер отвечает за организацию и проведение этого митинга. Команда помогает ему составить адженду и распланировать кто и в какой последовательности что представляет.

Подготовка к митингу не должна занимать у команды много времени (правило - не более двух часов). В частности, именно поэтому запрещается использовать презентации в Power Point. Подготовка к митингу не должна занимать у команды более 2-х часов.

6.4. Адаптивная разработка (ASD) по Хайсмиту

Следующая иллюстрация относится к области быстрых методологий, в которой заостряется внимание на необходимости адаптивной разработки. Основу ASD составляют три нелинейные, перекрывающие друг друга фазы: обдумывание, сотрудничество и обучение, относящиеся к каждому периоду разработки, который завершается выпуском релиза (Рисунок 2). Подчеркивается, что планирование в окружении, которое требует адаптивности, является парадоксом, поскольку результаты в этом случае всегда непредсказуемы. При обычном планировании отклонения от плана являются ошибками, которые нужно исправлять. В адаптивных разработках отклонения ведут к решениям, которые объективно обусловлены, а потому их следует считать правильными.

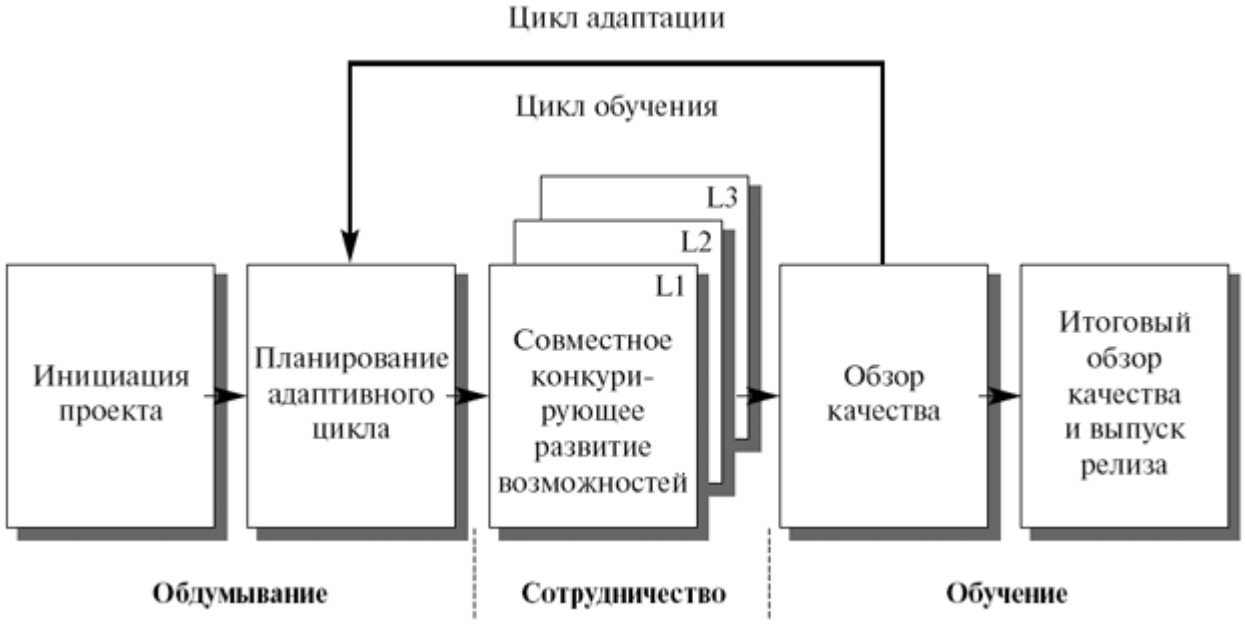


Рисунок 2. Модель жизненного цикла адаптивной разработки (ASD)

Неопределенность в столь непредсказуемой среде преодолевается за счет активного сотрудничества разработчиков. При этом внимание руководства направлено не столько на объяснения, что именно нужно делать, сколько на обеспечение коммуникации, при которой разработчики сами находят ответы на возникающие вопросы. Отсюда следует повышенное внимание к обучению, значение которого в предсказуемых методологиях часто занижается: все расписывается заранее, так что потом остается только следовать плану. Хайсмит пишет, что «в адаптивном окружении обучения не избежать всем участникам проекта – и разработчикам, и их заказчикам, поскольку и те и другие в процессе работы должны пересматривать собственные обязательства, а также использовать итоги каждого цикла разработки для того, чтобы подготовиться к следующему».

Таким образом, можно сказать, что ASD – это не готовая методология, а базовая концепция для различных адаптивных разработок. Схема жизненного цикла не является сдерживающим фактором построения конкретных методологий, которые вполне могут

следовать самым разнообразным стратегиям, включать в себя те или иные методики. В этом отношении подход Хайсмита сближается еще с одной «серийной» методологией быстрого развития, предложенной А. Коуберном. Речь идет о семействе методологий Crystal. Коуберн называет это семейством, так как убежден, что разным проектам нужны разные методологии. Он вводит следующую градацию проектов: по одной оси откладывается количество занятых в проекте людей, по другой – критичность ошибок. Каждая из методологий семейства предназначена для определенной ячейки получившейся сетки. Таким образом, проект, в котором занято 40 человек, и на котором компания может позволить себе потерять некоторую сумму, будет работать по другой методологии, нежели проект для шести разработчиков, от которого зависит существование компании.

Лекция 7. Методология сервис-менеджмента: ITSM, ITIL

7.1 ITSM

ITSM (IT Service Management, управление ИТ-услугами) – подход к управлению и организации ИТ-услуг, направленный на удовлетворение потребностей бизнеса. Управление ИТ-услугами реализуется поставщиками ИТ-услуг путём использования оптимального сочетания людей, процессов и информационных технологий

Проблема управления ИТ-ресурсами и повышения эффективности ИТ-услуг стара, как и само применение этих ресурсов и услуг. Поэтому сейчас, говоря об ITSM, мы имеем в виду новые концептуальные подходы к решению тех вопросов, которые были на теоретическом уровне сформулированы 20 лет назад и оказались реально востребованы фактически только сейчас, в начале нового столетия.

Ключевая идея ITSM в современном ее понимании заключается в необходимости перехода от традиционной модели, где главная цель - это собственно поддержка ИТ-инфраструктуры, к схеме, ориентированной на обслуживание основного бизнеса компании. Решение такой задачи осложняется тем, что для этого потребуется довольно радикально пересмотреть общее позиционирование сервисных ИТ-подразделений в структуре компаний. И тут нужно обратить внимание на две стороны данного вопроса.

Во-первых, ИТ-инфраструктура предприятий зачастую формировалась весьма хаотичным образом, оперативно отвечая на те или иные запросы со стороны основного бизнеса. В результате ИТ-службы обычно представляют собой весьма запутанную структуру, как с технической, так и экономической точки зрения.

Во-вторых, ИТ-департаменты исторически рассматриваются как вспомогательные, сугубо бюджетные подразделения. А это означает (точнее, из этого следует), что руководство компаний не может четко выявить взаимосвязь между инвестициями в развитие и поддержку ИС и повышением эффективности основного бизнеса.

Повышение интереса к концепции ITSM в значительной степени определялось также экономическим кризисом начала нынешнего века, когда во многих компаниях - довольно неожиданно для себя - наряду с дефицитом выделяемых им бюджетов стали ощущать новые требования со стороны руководства в виде необходимости предоставления отчетов по расходам и сведений об ожидаемой прибыли от инвестиций в ИТ-ресурсы. Это подтверждается целым рядом исследований по всему миру. Их результаты говорят также о том, что ИТ-менеджеры не всегда могут четко определить, какие преимущества получают внутренние или внешние клиенты ИТ-подразделений от той или иной услуги.

По оценкам Meta Group, ситуация на рынке такова, что около 75% ИТ-подразделений сегодня выступает в роли не более чем поставщиков инфраструктуры, ориентированных исключительно на ее технологическое развитие вне связи с деятельностью предприятий в целом. В то же время компании хотят пользоваться экономически эффективными ИТ-услугами, отвечающими их индивидуальным потребностям и способными помочь им в решении ключевых бизнес-задач. Поэтому ИТ-департаменты должны предпринять усилия и сделать шаг вперед, который позволит им стать не просто поставщиками ИТ-инфраструктуры, а настоящими сервис-провайдерами, а затем и стратегическими партнерами руководства компаний, предоставляющими широкий спектр услуг, эффективность которых поддается достаточно простой оценке со стороны их потребителей.

Meta Group считает, что во всем мире только 25% компаний приступило к внедрению сервисной модели обслуживания и лишь 5% из них удалось вырасти до того уровня (по состоянию на 2003 г.), когда ИТ-подразделение становится для своей компании ценным

стратегическим ресурсом. Однако по прогнозам, приведенным в том же отчете, в ближайшие три года такой переход сможет осуществить подавляющее большинство ИТ-подразделений крупных и даже средних компаний.

По сути дела, концепция ITSM полностью соответствует общей нацеленности заказчиков на более широкое использование ИТ-аутсорсинга, в том числе и в сфере услуг. Таким образом, задачей ИТ-подразделений становится применение модели аутсорсинга на внутреннем уровне своей организации. И в этой связи аналитики предупреждают: тех, кому не удастся успешно реализовать новые принципы работы, скорее всего ждет расформирование, а их функции будут возложены на внешние специализированные организации.

7.2 ITIL

В настоящее время стандартом «де-факто» в области организации и управления информационными технологиями, обобщившим в себе лучший международный опыт, является Information Technology Infrastructure Library (ITIL).

Издателем Библиотеки ITIL является OGC (The Office of Government Commerce) - британская правительственная организация, отвечающая за повышение эффективности работы государственных структур Великобритании, а также за развитие кооперации с компаниями частного сектора.

Развитие и популяризация Библиотеки поддерживаются не только её издателем (OGC) но и независимым профессиональным сообществом itSMF (IT Service Management Forum). Эта некоммерческая организация объединяет как частных лиц, профессионалов в области управления ИТ, так и организации, в том числе компании-вендоры. В числе крупнейших корпоративных членов Форума (Global members) компании Microsoft, SUN, HP и IBM.

Библиотека ITIL первоначально была результатом работы Центрального агентства по вычислительной технике и телекоммуникациям (Central Computer and Telecommunications Agency – CCTA) при правительстве Великобритании. В апреле 2001 г. CCTA было объединено с Государственной торговой палатой (Office of Government Commerce – OGC), являющейся в настоящее время новым владельцем библиотеки ITIL. Целью OGC является помощь заказчикам из государственного сектора экономики Великобритании в модернизации их деятельности по закупкам и улучшении обслуживания путем максимального использования ИТ и других инструментариев. «Задачей OGC является модернизация закупок правительственными службами и работа по улучшению использования финансовых средств». OGC способствует использованию «передового опыта» в различных сферах деятельности (например, Управление Проектами, Управление Закупками и ИТ Сервис-менеджмент). OGC выпускает несколько книжных серий (библиотек), написанных экспертами из Великобритании и других стран мира, представляющими ряд компаний и организаций.

Принадлежащая OGC библиотека ITIL состоит из ряда доступных и детальных «Собраний практических руководств» для предоставления эффективных и рациональных ИТ-услуг (сервисов).

Книги библиотеки ITIL

Каждая из книг библиотеки ITIL рассматривает вопросы отдельной части структурированной процессной основы. В них дается описание того, что необходимо для организации ИТ Сервис-менеджмента.

Библиотека ITIL определяет цели и виды деятельности, входные и выходные параметры каждого из процессов в ИТ-организации. Однако, библиотека ITIL не дает конкретного описания способов осуществления этой деятельности, так как они могут различаться в каждой организации. Акцент делается на проверенном практикой подходе,

который может быть реализован различными способами в зависимости от обстоятельств. Библиотека ITIL не является методом; наоборот, она предлагает структурированную основу для планирования наиболее часто используемых процессов, ролей и видов деятельности, определяя связи между ними и необходимые виды коммуникации.

Создание библиотеки ITIL вызвано необходимостью предоставления высококачественных услуг, уделяя при этом особое внимание отношениям с заказчиком. ИТ-организация должна выполнять соглашения с заказчиком, что означает поддержание хороших отношений с заказчиками и партнерами, такими, как поставщики.

Частично философия библиотеки ITIL имеет в своей основе системы качества, такие как стандарты серии ISO-9000, и общие схемы обеспечения качества (Total Quality frameworks), как предлагаемые Европейской организацией Управления Качеством (European Foundation of Quality Management – EFQM). Библиотека ITIL поддерживает эти системы путем предоставления четкого описания процессов и передового опыта ИТ Сервис-менеджмента. Это может значительно сократить время, необходимое для прохождения сертификации ISO.

Первоначально библиотека ITIL состояла из нескольких комплектов книг, в каждом из которых описывалась конкретная область сопровождения и эксплуатации ИТ-инфраструктуры. Ядром ITIL считались десять книг, в которых описывались такие области, как поддержка услуг и предоставление услуг. Библиотека включала также около 40 других книг по вспомогательным предметам, имевшим отношение к ИТ Сервис-менеджменту, от монтажа кабелей до Управления Отношениями с Заказчиком. Для управления книгами используется book manager. Однако, в первоначальных сериях книг вопросы ИТ Сервис-менеджмента рассматривали главным образом с точки зрения ИТ. Для заполнения разрыва между бизнес-практикой и ИТ-организацией в библиотеку была включена серия книг, рассматривающая бизнес-аспекты ИТ Сервис-менеджмента. Более того, в определенных частях библиотеки ITIL в то время использовался несколько устаревший подход.

В настоящее время была сделана переработка центральных книг по ITIL и они были переизданы в виде двух книг: одна – по поддержке услуг и другая – по их предоставлению. Это позволило исключить повторы и встречавшуюся местами несогласованность ранних серий, что улучшило структурное единство издания, которое теперь дает более четкое представление об ИТ Сервис-менеджменте.

Начатый после выпуска указанного издания пересмотр всей серии публикаций ITIL еще не завершен.

В настоящее время ITIL состоит из восьми томов:

1) Service Support
2) Service Delivery
3) Planning to Implement Service Management
4) Application Management
5) ICT Infrastructure Management
6) Security Management
7) Software Asset Management
8) The Business Perspective: The IS View on Delivering Services to the Business

Парадигма ITIL

- Задача ИТ-службы – обеспечение основного бизнеса всеобъемлющим набором информационных сервисов.

- Соглашение об уровне предоставления сервисов (Service Level Agreement) – согласованный и утвержденный документ, на основании которого сервисы поставляются бизнесу.
- Качество сервиса – измеряемая величина.
- Деятельность ИТ-службы организуется на основании процессного подхода

Преимущества ITIL

- Использование лучших знаний и проверенных практик;
- Ориентация работы IT на решение задач бизнеса;
- IT службы представляются поставщиками IT-услуг для бизнес подразделений;
- Деятельность IT регламентируется соглашением об уровне услуг;
- Определение стандартов и правил для IT-персонала;
- Нацеленность на обеспечение максимально возможного качества IT-услуг для пользователей;
- Внедрение подходов менеджмента качества в управления IT-сервисами;
- Возможность подтвердить и объяснить стоимость IT в соответствии с согласованным уровнем обслуживания.

Библиотека ITIL постоянно перерабатывается и обновляется, аккумулируя новый опыт и новые знания. Сегодня рекомендациями и методологиями ITIL пользуются большинство крупнейших компаний по всему миру, повышая эффективность управления ИТ инфраструктурой.

Лекция 8. Стадии жизненного цикла ИТ-проекта

8.1 Стадии жизненного цикла ИТ-проекта. Руководство PMBOK

Все эти пятьдесят лет методика управления проектами не стояла на месте, а постоянно развивалась. За это время появился целый ряд национальных и международных организаций, которые занимаются разработкой и поддержанием стандартов.

Данные стандарты представляют собой свод знаний об управлении проектами (Project Management Body of Knowledge – PMBOK). Но структура и содержание PMBOK в разных странах может отличаться в связи с тем, что многие национальные ассоциации управления проектами имеют неодинаковые точки зрения на то, что именно должно входить в этот документ.

Функциональная структура управления проектами включает в себя девять разделов.

1) Управление координацией (Project Integration Management).
2) Управление целями (Project Scope Management).
3) Управление временем (Project Time Management).
4) Управление стоимостью (Project Cost Management).
5) Управление качеством (Project Quality Management).
6) Управление человеческими ресурсами (Project Human Resource Management).
7) Управление коммуникациями (Project Communication Management).
8) Управление рисками (Project Risk Management).
9) Управление поставками (Project Procurement Management).

Все эти функции тесно переплетены между собой.

В каждом проекте (фазе проекта) обязательно присутствуют пять групп процессов.

Группы процессов

Процессы управления проектами могут быть разбиты на шесть основных групп, реализующих различные *функции* управления:

1) **процессы инициации** - принятие решения о начале выполнения проекта;
2) **процессы планирования** - определение целей и критериев успеха проекта и разработка рабочих схем их достижения;
3) **процессы исполнения** - координация людей и других ресурсов для выполнения плана;
4) **процессы анализа** - определение соответствия плана и исполнения проекта поставленным целям и критериям успеха и принятие решений о необходимости применения корректирующих воздействий;
5) **процессы управления** - определение необходимых корректирующих воздействий, их согласование, утверждение и применение;
6) **процессы завершения** - формализация выполнения проекта и подведение его к упорядоченному финалу.

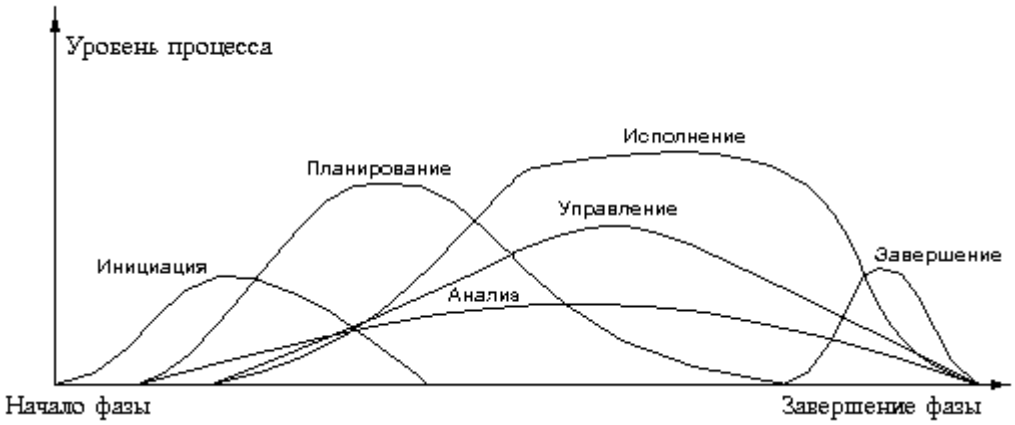


Рисунок 1. Наложение групп процессов в фазе

Процессы управления проектами накладываются друг на друга и происходят с разными интенсивностями на всех стадиях проекта, как проиллюстрировано на Рисунок 1.

Кроме того, процессы управления проектами связаны своими результатами - результат выполнения одного становится исходной информацией для другого.

И, наконец, имеются взаимосвязи групп процессов различных фаз проекта. Например, закрытие одной фазы может являться входом для инициации следующей фазы (пример: завершение фазы проектирования требует одобрения заказчиком проектной документации, которая необходима для начала реализации). В реальном проекте фазы могут не только предшествовать друг другу, но и накладываться. Повторение инициации на разных фазах проекта помогает контролировать актуальность выполнения проекта. Если необходимость его осуществления отпала, очередная инициация позволяет вовремя это установить и избежать излишних затрат.

Взаимосвязи процессов

Внутри каждой группы процессы управления проектами связаны друг с другом через свои входы и выходы. Фокусируясь на этих связях, опишем отдельные процессы через:

- **Входы** – документы или документированные показатели, согласно которым процесс исполняется.
- **Выходы** – документы или документированные показатели, являющиеся результатом процесса.
- **Методы и средства** – механизмы, по которым вход преобразуется в выход.

Описываемые ниже процессы характерны для большинства проектов и подробнее освещены в последующих главах.

1. Процессы инициации

Инициация включает единственный подпроцесс - *Авторизацию*, т.е. решение начать следующую фазу проекта.

2. Процессы планирования

Планирование имеет большое значение для проекта, поскольку проект содержит то, что ранее не выполнялось. Естественно, что планирование включает сравнительно много процессов. Однако не следует считать, что управление проектами это в основном планирование. Усилия, прилагаемые для планирования, следует соизмерять с целями проекта и полезностью полученной информации.

Напомним, что следует различать цели проекта и цели продукта проекта, под которым понимается продукция (или услуги), созданная или произведенная в результате исполнения проекта.

Цели продукта – это свойства и функции, которыми должна обладать продукция проекта.

Цели проекта – это работа, которую нужно выполнить для производства продукта с заданными свойствами.

В ходе исполнения проекта эти процессы многократно повторяются. Изменениям могут подвергнуться цели проекта, его бюджет, ресурсы и т.д. Кроме того, планирование проекта - это не точная наука. Различные команды проекта могут разработать различные планы для одного и того же проекта. А пакеты управления проектами могут составить различные расписания выполнения работ при одних и тех же исходных данных.

Основные процессы планирования

Некоторые из процессов планирования имеют четкие логические и информационные взаимосвязи и выполняются в одном порядке практически во всех проектах. Так, например, сначала следует определить из каких работ состоит проект, а уж затем рассчитывать сроки выполнения и стоимость проекта. Эти основные процессы выполняются по несколько раз на протяжении каждой фазы проекта. К основным процессам планирования относятся:

- *планирование целей* – разработка постановки задачи (проектное обоснование, основные этапы и цели проекта);
- *декомпозиция целей* – декомпозиция этапов проекта на более мелкие и более управляемые компоненты для обеспечения более действенного контроля;
- *определение состава операций* (работ) проекта – составление перечня операций, из которых состоит выполнение различных этапов проекта;
- *определение взаимосвязей операций* – составление и документирование технологических взаимосвязей между операциями;
- *оценка длительностей или объемов работ* – оценка количества рабочих временных интервалов, либо объемов работ, необходимых для завершения отдельных операций;
- *определение ресурсов* (людей, оборудования, материалов) проекта – определение общего количества ресурсов всех видов, которые могут быть использованы на работах проекта (ресурсов организации) и их характеристик;
- *назначение ресурсов* – определение ресурсов, необходимых для выполнения отдельных операций проекта;
- *оценка стоимостей* – определение составляющих стоимостей операций проекта и оценка этих составляющих для каждой операции, ресурса и назначения;
- *составление расписания выполнения работ* – определение последовательности выполнения работ проекта, длительностей операций и распределения во времени потребностей в ресурсах и затрат, исходя и с учетом наложенных ограничений и взаимосвязей;
- *оценка бюджета* – приложение оценок стоимости к отдельным компонентам проекта (этапам, фазам, срокам);
- *разработка плана исполнения проекта* – интеграция результатов остальных подпроцессов для составления полного документа;

– *определение критериев успеха* – разработка критериев оценки исполнения проекта.

Вспомогательные процессы планирования

Кроме перечисленных основных процессов планирования имеется ряд вспомогательных процессов, необходимость в использовании которых сильно зависит от природы конкретного проекта. Такие процессы включают в себя:

– *планирование качества* – определение того, какие стандарты качества использовать в проекте, и того, как эти стандарты достичь;

– *планирование организации* – определение, документирование и назначение ролей, ответственности и взаимоотношений отчетности в организации;

– *назначение персонала* – назначение человеческих ресурсов на выполнение работ проекта;

– *планирование взаимодействия* – определение потоков информации и способов взаимодействия, необходимых для участников проекта;

– *идентификация риска* – определение и документирование событий риска, которые могут повлиять на проект;

– *оценка риска* – оценка вероятностей наступления событий риска, их характеристик и влияния на проект;

– *разработка реагирования* – определение необходимых действий для предупреждения рисков и реакции на угрожающие события;

– *планирование поставок* – определение того, что, как и когда должно быть поставлено;

– *подготовка условий* – выработка требований к поставкам и определение потенциальных поставщиков.

Взаимосвязи между вспомогательными подпроцессами, как и само их наличие, в большой мере зависят от природы проекта.

3. Процессы исполнения и контроля

Под исполнением подразумеваются процессы реализации составленного плана. Исполнение проекта должно регулярно измеряться и анализироваться для того, чтобы выявить отклонения от намеченного плана и оценить их влияние на проект. Регулярное измерение параметров проекта и идентификация возникающих отклонений далее также относится к процессам исполнения и именуется *контролем исполнения*. Контроль исполнения следует проводить по всем параметрам, входящим в план проекта.

Как и в планировании, процессы исполнения можно подразделить на основные и вспомогательные.

К ***основным*** можно отнести сам процесс *исполнения* плана проекта.

Среди ***вспомогательных процессов*** отметим:

– *учет исполнения* – подготовка и распределение необходимой для участников проекта информации с требуемой периодичностью;

– *подтверждение качества* – регулярная оценка исполнения проекта с целью подтверждения соответствия принятым стандартам качества;

– *подготовка предложений* – сбор рекомендаций, отзывов, предложений, заявок и т.д.;

- *выбор поставщиков* – оценка предложений, выбор поставщиков и подрядчиков и заключение контрактов;
- *контроль контрактов* – контроль исполнения контрактов поставщиками и подрядчиками;
- *развитие команды проекта* – повышение квалификации участников команды проекта.

4. Процессы анализа

Процессы анализа включают как анализ плана, так и анализ исполнения проекта.

Анализ плана означает определение того, удовлетворяет ли составленный план исполнения проекта предъявляемым к проекту требованиям и ожиданиям участников проекта. Он выражается в оценке показателей плана командой и другими участниками проекта. На стадии планирования результатом анализа плана может быть принятие решения о необходимости изменения начальных условий и составления новой версии плана, либо принятие разработанной версии в качестве базового плана проекта, который в дальнейшем служит основой для измерения исполнения. В дальнейшем изложении анализ плана не выделяется в качестве отдельной группы процессов, а включается в группу процессов планирования, делая эту группу процессов по своей природе итеративной. Таким образом, под процессами анализа в дальнейшем понимаются процессы анализа исполнения.

Процессы анализа исполнения предназначены для оценки состояния и прогноза успешности исполнения проекта согласно критериям и ограничениям, определенным на стадии планирования. В силу уникальности проектов эти критерии не являются универсальными, но для большинства проектов в число основных ограничений и критериев успеха входят цели, сроки, качество и стоимость работ проекта. При отрицательном прогнозе принимается решение о необходимости корректирующих воздействий, выбор которых осуществляется в процессах управления изменениями.

Процессы анализа также можно подразделить на основные и вспомогательные.

К ***основным*** относятся те процессы анализа, которые непосредственно связаны с целями проекта и показателями, характеризующими успешность исполнения проекта:

- *анализ сроков* – определение соответствия фактических и прогнозных сроков исполнения операций проекта директивным или запланированным;
- *анализ стоимости* – определение соответствия фактической и прогнозной стоимости операций и фаз проекта директивным или запланированным;
- *анализ качества* – мониторинг результатов с целью их проверки на соответствие принятым стандартам качества и определения путей устранения причин нежелательных результатов исполнения качества проекта;
- *подтверждение целей* – процесс формальной приемки результатов проекта его участниками (инвесторами, потребителями и т.д.).

Вспомогательные процессы анализа связаны с анализом факторов, влияющих на цели и критерии успеха проекта. Эти процессы включают:

- *оценку исполнения* – анализ результатов работы и распределение проектной информации с целью снабжения участников проекта данными о том, как используются ресурсы для достижения целей проекта;
- *анализ ресурсов* – определение соответствия фактической и прогнозной загрузки и производительности ресурсов запланированным, а также анализ соответствия фактического расхода материалов плановым значениям.

В число процессов анализа не включены анализ взаимодействия с целью оптимизации процедур обработки проектной информации, анализ исполнения контрактов с целью своевременного внесения изменений и предотвращения споров и ряд других процессов, которые не носят регулярного характера (как анализ взаимодействия), либо составляют часть включенных процессов (как анализ контрактов).

В результате анализа либо принимается решение о продолжении исполнения проекта по намеченному ранее плану, либо определяется необходимость применения корректирующих воздействий

5. Процессы управления

Управление исполнением проекта – это определение и применение необходимых управляющих воздействий с целью успешной реализации проекта. Если исполнение проекта происходит в соответствии с намеченным планом, то управление фактически сводится к исполнению - доведению до участников проекта плановых заданий и контролю их реализации. Эти процессы нами включены в процессы исполнения. Другое дело, если в процессе реализации возникли отклонения, анализ которых показал, что необходимо определение и применение корректирующих воздействий. В этом случае требуется найти оптимальные корректирующие воздействия, скорректировать план оставшихся работ и согласовать намеченные изменения со всеми участниками проекта.

Итак, процессы управления предназначаются для определения, согласования и внесения необходимых изменений в план проекта. Такие процессы управления часто называются управлением изменениями и инициируются процессами анализа.

К ***основным процессам*** управления, встречающимся практически в каждом проекте, относятся:

- *общее управление изменениями* – определение, согласование, утверждение и принятие к исполнению корректирующих воздействий и координация изменений по всему проекту.
- *управление ресурсами* – внесение изменений в состав и назначения ресурсов на работы проекта;
- *управление целями* – корректировка целей проекта по результатам процессов анализа;
- *управление качеством* – разработка мероприятий по устранению причин неудовлетворительного исполнения.

Среди ***вспомогательных процессов*** управления отметим:

- *управление рисками* – реагирование на события и изменение рисков в процессе исполнения проекта;
- *управление контрактами* – координация работы (суб)подрядчиков, корректировка контрактов, разрешение конфликтов.

6. Процессы завершения

Завершение проекта сопровождается следующими процессами:

- *закрытие контрактов* – завершение и закрытие контрактов, включая разрешение всех возникших споров.
- *административное завершение* – подготовка, сбор и распределение информации, необходимой для формального завершения проекта.

8.2 Управление рисками: анализ, планирование, мониторинг и контроль

Управляя рисками, ИТ-подразделение стремится нейтрализовать их и при этом достичь целей управления. Долгосрочный успех в управлении рисками достигается за счет эффективного использования внутренних мер контроля.

Процесс идентификации рисков и мер контроля затрагивает все компоненты компании. Он обеспечивает основу для корпоративных усилий по соблюдению нормативных требований, четко определяя взаимосвязи между целями, возможными препятствиями на пути их достижения и способами воздействия на эти препятствия.

Каждому этапу жизненного цикла ИТ-услуги соответствует набор типовых рисков и действий по управлению ими.

- На этапе «Планирование» в центре внимания находятся риски, связанные с конкретными стратегиями и информационными архитектурами, а также риски в портфеле ИТ-услуг.
- На этапе «Внедрение» риски оцениваются с точки зрения проектов, т. е. более целенаправленно и ограниченно по времени.
- Этап «Эксплуатация» сосредоточен на текущей деятельности и рисках, которые могут повлиять на надежность эксплуатации.
- В завершение на уровне «Управление» выполняется как общее, так и специализированное управление рисками: общее — в части структуры управления, организационной координации, принятия решений и планов коммуникации; специализированное — в части управления изменениями, конфигурациями и рисками, возникающими при изменении элементов среды ИТ-услуг, а также процессов и ресурсов, являющихся частью этой среды).

На каждом этапе жизненного цикла ИТ-услуги присутствуют разные категории рисков. В их числе риски, связанные с финансами, эксплуатацией, репутацией, рыночной долей, доходом и нормативными требованиями, а также риски, обусловленные отраслевой принадлежностью организации (например, здравоохранение) или происходящими в данный момент событиями (например, слияние или поглощение).

Правильное управление рисками, которое заставляет думать о возможных последствиях действий, оценивать их влияние, а затем использовать очень четкий подход к нейтрализации связанных с ними рисков, дает ИТ-подразделению значительные преимущества. Организация не сможет разумно противостоять рискам, если ИТ и бизнес совместно не определят допустимость рисков и меры их контроля. Поскольку последствия рисков оцениваются с точки зрения достижения бизнес-целей, это позволяет ИТ и бизнесу совместно обсуждать риски и принимать компромиссные решения, а также избежать перекладывания вины за ошибки друг на друга благодаря прозрачности процесса управления рисками.

Этот процесс включает в себя следующие действия:

- усовершенствование процессов с учетом целей управления;
- идентификация рисков;
- анализ и приоритизация рисков;
- идентификация мер контроля;
- анализ мер контроля;
- планирование и составление графика внедрения;

- отслеживание рисков и мер контроля и составление отчетов по ним;
- использование мер контроля;
- изучение предшествующего опыта и обновление базы знаний.

8.3 Документация ИТ-проекта

При реализации проекта всегда разрабатываются три основных документа: техническое задание, план-график и бюджет. Они позволяют успешно организовать работу, но только в том случае, если составлены специалистом.

Техническое задание (ТЗ) – основной документ проекта, в соответствии с которым проводится разработка и введение в действие системы. В нем перечисляются задачи по автоматизации и способы их решения, включая конкретные описания форм, отчетов, справочников, характеристика объектов автоматизации, требования к системе, состав и содержание работ по ее созданию, алгоритмы взаимодействия всех элементов, а также порядок контроля и приемки системы. Разрабатывает документ менеджер проекта совместно с ИТ-службой по определенному стандарту (ГОСТ на ТЗ по созданию автоматизированной системы). Правильно составленное техническое задание – это залог успешного взаимодействия всех участников проекта. Чем точнее оно написано, тем быстрее и эффективнее будут выполняться работы.

К настоящему времени уже сформированы стандартные требования к автоматизированным системам:

- интегрируемость с различными информационными продуктами, используемыми в компании;
- надежность – обеспечение устойчивости работы и хранения данных;
- гибкость – способность легко адаптироваться к любым изменениям, происходящим в организации;
- масштабируемость – возможность увеличения объемов обрабатываемой информации и количества пользователей;
- возможность развития функционала фирмой-разработчиком в ходе эксплуатации системы – обновление версий, актуализация под требования российского законодательства и правовых норм;
- возможность работы с распределенными структурами (удаленными подразделениями);
- распределение прав доступа к информации в соответствии с должностями пользователей.

План-график обычно содержит стандартные для любого проекта организационно-технические мероприятия: оценка трудозатрат, подготовка рабочих мест и технической документации, формирование команды и ее обучение, тестирование системы, ее запуск в опытную и, затем, промышленную эксплуатацию. Менеджер проекта указывает в плане-графике сроки и исполнителей по каждому виду работ.

Бюджет ИТ-проекта включает следующие основные статьи затрат: покупка программного обеспечения, настройка, сопровождение. В зависимости от масштабов проекта и выбранной технологии соотношение расходов по этим статьям может быть разным.

Лекция 9. Организация проектной команды

9.1 Организация проектной команды

При распределении ролей и ответственности, необходимых для выполнения проекта, следует учитывать следующие моменты.

Роль в проекте (проектная роль) – определенный набор функций и полномочий в проекте, созданный с целью распределения обязанностей между членами команды проекта. Проектную роль можно рассматривать как временную должность в организации (компании).

Полномочия – право задействовать ресурсы проекта, принимать решения и утверждать одобрение действий или результатов. Примеры полномочий: выбор способа завершения операции, приемка качества и порядок реагирования на отклонения в проекте.

Ответственность – работа, которую член команды проекта должен выполнить для завершения операций проекта.

Квалификация – навыки и способности, необходимые для выполнения операций проекта. Отсутствие нужной квалификации у членов команды влияет на расписание проекта, качество выполнения работ, ставит под угрозу цели проекта. Для повышения квалификации планируют проведение обучения членов команды.

Формируя команду управления проектом, необходимо определить ключевых лиц проекта, принимающих решения.

Со стороны заказчика ключевые роли играют спонсор проекта и менеджер проекта со стороны заказчика. Спонсор проекта обеспечивает организационную сторону проекта и подтверждает правильность целей проекта. В его ведении находится бюджет проекта. Спонсором проекта может быть отдельный человек или целый комитет, в зависимости от масштабов и сложности проекта. Менеджер проекта со стороны заказчика назначается и в том случае, если осуществление проекта организацией заказчика требует ежедневного управления. В его обязанности входит предоставление ресурсов заказчиков, разрешение проблем и отслеживание состояния проекта.

Ключевые роли со стороны исполнителя - руководитель проекта (менеджер проекта) со стороны исполнителя и бизнес-менеджер.

Бизнес-менеджер отвечает за успешное выполнение проекта и представляет исполнителя в его договорных отношениях с заказчиком. Менеджер проекта (руководитель проекта) отвечает как за успехи, так и за неудачи проекта. В его задачи входит управление сроками, стоимостью, качеством работ с целью удовлетворения ожиданий заказчика и достижения бизнес-целей исполнителя.

Команда управления проектом включает координатора проекта, администратора проекта, менеджера по конфигурации. Для крупных проектов к выполнению каждой из этих ролей могут быть привлечено нескольких человек. На небольших проектах менеджер проекта может совмещать несколько ролей. Масштабные проекты предполагают наличие менеджера по качеству, который ответственен перед бизнес-менеджером исполнителя.

Приведенный список ключевых ролей команды управления проектом является необходимым для управления работами при внедрении информационной системы. Возможны некоторые модификации состава команды в зависимости от сложности и масштабности проекта, например, при необходимости можно включать в нее заместителя руководителя проекта, руководителей функциональных направлений (финансы, логистика, персонал и т. д.).

Состав команды управления должен быть достаточным, чтобы осуществлять:

- управление ресурсами проекта, в том числе:
    - определение требуемых для достижения целей проекта ресурсов;
    - подготовка предложений по изменению состава группы управления проектом;
    - утверждение персональных изменений в составе рабочих групп проекта;
    - оценка стоимости проекта, подготовка бюджетов проекта и отчетов об исполнении бюджетов;
- управление сроками выполнения проекта, в том числе:
    - подготовка плана работ проекта;
    - контроль над выполнением проекта;
    - подготовка отчетов о ходе работ проекта;
- управление качеством проекта, в том числе:
    - контроль соответствия разрабатываемых проектных решений техническому заданию;
    - организация экспертизы проектных решений;
- управление рисками проекта, в том числе:
    - анализ рисков проекта;
    - разработка планов мероприятий по снижению рисков;
    - реализация мероприятий по снижению рисков;
- управление проблемами проекта, в том числе:
    - анализ проблем проекта;
    - разработка мероприятий по разрешению проблем проекта;
    - реализация мероприятий по разрешению проблем проекта;
- контроль над организацией работ в проектных группах, в том числе:
    - согласование отчетов о ходе работ;
    - контроль над функционированием системы сбора и распределения информации;
    - контроль документирования проектных результатов.

В состав команды проекта входят не только команда управления проектом, но и исполнители проекта. Примеры проектных ролей исполнителей, характерных для IT-проектов: функциональный архитектор, функциональный консультант, разработчик, администратор ИС, тестировщик, менеджер по качеству, системный аналитик. В проекте один член команды может выступать одновременно в нескольких ролях. Совмещение ролей часто встречается в небольших проектах, что позволяет снизить накладные расходы проекта. Но не все роли можно совмещать, поскольку подобное совмещение может затруднить контроль и оценку результатов проекта. Допускается совмещение таких проектных ролей, как руководитель проекта и администратор проекта, функциональный архитектор и функциональный консультант, функциональный консультант и аналитик, менеджер разработки и разработчик, менеджер по качеству и тестировщик. Но не следует совмещать роли менеджера по качеству и разработчика, руководителя проекта и разработчика, тестировщика и разработчика.

На стадии планирования в рамках процесса управления человеческими ресурсами не предусматривается долгосрочное планирование, а составляется план для реализации первого

этапа проекта. Основными задачами являются разработка организационной структуры проекта и подбор персонала.

Работа по планированию организационной структуры проводится менеджером проекта со стороны исполнителя совместно с менеджером со стороны заказчика. Путем переговоров достигается соглашение об уровне, на котором должно производиться утверждение выделяемых ресурсов заказчика и обсуждение требований к членам команды исполнителя. Администратор проекта фиксирует результаты переговоров.

Иерархические организационные диаграммы являются простым и наглядным инструментом для определения иерархии подотчетности, начиная с нижнего уровня организации до руководителя проекта.

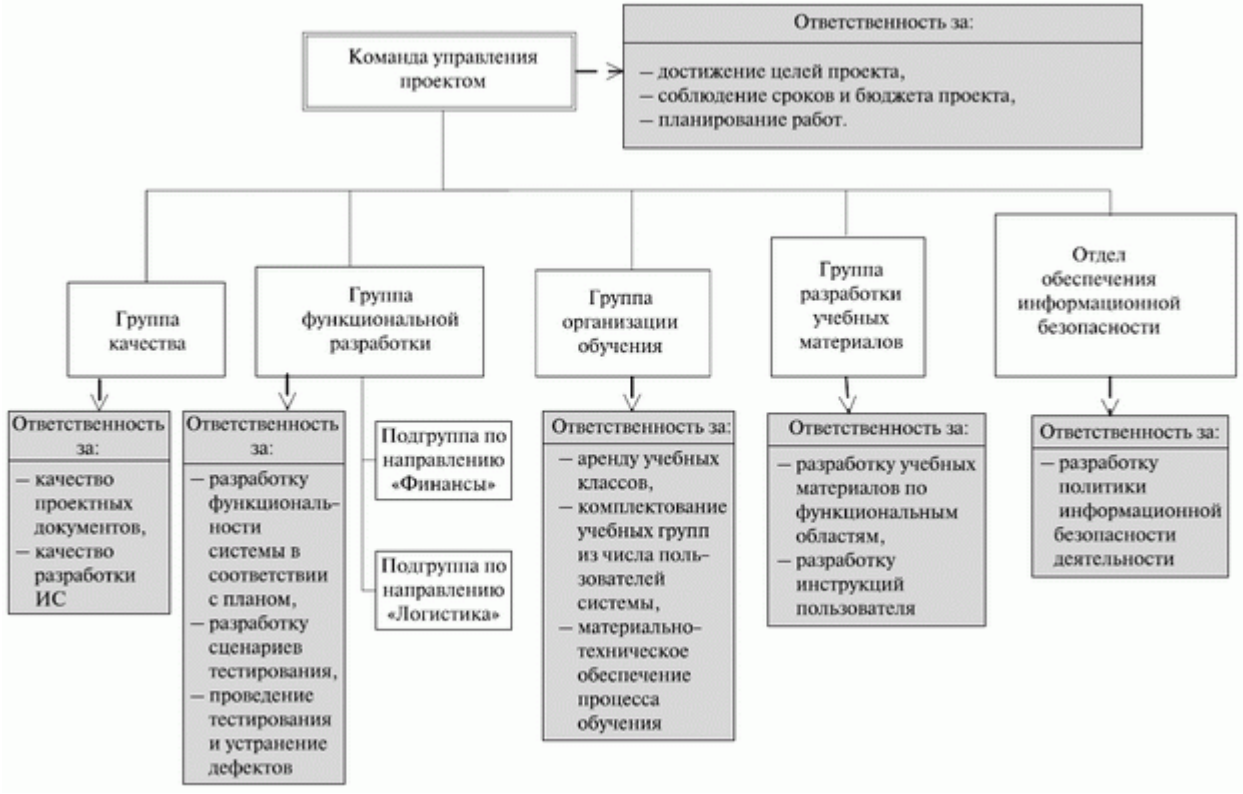


Рисунок 1. Пример организационной структуры проекта

Существуют различные форматы документирования распределения ролей и ответственности членов команды проекта, например, иерархический, матричный или текстовый. Независимо от формата документирования организационные диаграммы позволяют для каждого пакета работ назначить ответственного за его исполнение, а также обеспечивают понимание своей роли и ответственности каждым членом команды.

9.2 Матрица ответственности проекта

Для отражения иерархии подотчетности на проекте и указания обязанностей каждой из групп, входящих в проектную команду, в документ описания содержания проекта рекомендуется включить матрицу ответственности, наиболее распространенный вариант которой известен как RACI-матрица. Использование данного инструмента особенно актуально в ситуации, когда проектная команда состоит из представителей различных юридических лиц. Матрица ответственности решает задачу демонстрации межорганизационного или межгруппового взаимодействия и, как следствие, позволяет избежать недоразумений, которые

время от времени возникают в проектах между подразделениями и организациями из-за неясности, к кому следует обращаться по тем или иным вопросам и кто должен принимать по ним решение, а кто - непосредственно реализовать принятую резолюцию.

Важно как можно раньше произвести размежевание всех формальных полномочий, прав и обязанностей, пока команда проекта еще не приступила к активной работе. В противном случае, когда у сотрудников сложится собственное представление о своем месте в проекте, расхождения во мнениях по этим вопросам могут перерасти в затяжные конфликты и оказать значительное негативное влияние на график выполнения проекта.

Построение матрицы ответственности

1) Перечислить основные работы проекта.

По вертикали в матрице отражаются только основные работы проекта (не ниже уровня 2-3 ИСР), но с достаточной степенью детализации для обеспечения возможности указывать разные роли, необходимые для выполнения этих работ. Когда речь идет о крупных проектах и программах, может возникнуть необходимость разработать несколько матриц ответственности с различной степенью детализации.

2) Перечислить группы/роли внутри проектной команды.

По горизонтали в матрице перечисляются группы/ роли внутри проектной команды. Обратите внимание на то, что в матрице ответственности группы/роли, а не имена и фамилии отдельных членов коллектива. Персональное закрепление проектных работ производится позднее, на этапе разработки расписания проекта.

3) Закодировать матрицу ответственности.

С помощью кодов в ячейках на пересечении соответствующих столбцов с ролями и строк с работами проекта указать степень участия, формальные полномочия и распределение ответственности за выполнение каждой операции. Четкое указание разных уровней формальных полномочий бывает особенно полезно в ситуации, когда множество членов проектной команды желает предъявить особые требования к проекту.

На коды, используемые в матрице ответственности, каких-либо ограничений не существует, но наибольшее распространение получил метод RACI (Responsible (R), Accountable (A), Consulted (C), Informed(I)), в котором приведено описание соответствующих кодов.

Таблица 1. Условные обозначения матрицы ответственности (RACI)

Обозначение	**Расшифровка**	**Описание**
Исп. (R)	Исполнитель (Responsible)	Несет ответственность за непосредственное исполнение задачи. К каждой задаче должно быть приписано не менее одного исполнителя
Утв. (A)	Утверждающий (Accountable)	Отвечает за конечный результат перед вышестоящим руководством. На каждую работу должен быть назначен строго один подотчетный
Согл. (C)	Согласующий (Consulted)	Согласует принимаемые решения, взаимодействие с ним носит двусторонний характер
Н. (I)	Наблюдатель (Informed)	Его информируют об уже принятом решении, взаимодействие с ним носит односторонний характер

4) Инициировать использование матрицы и включить процедуру использования матрицы ответственности в документ "План управления проектом".

После утверждения матрицы ответственности все дальнейшие изменения в ней должны проходить через процедуру интегрированного управления изменениями при участии авторов первоначальной версии.

Преимущество использования структурированного подхода к изменению матрицы ответственности состоит в том, что руководитель проекта получает актуальный документ, на который он может ссылаться при возникновении тех или иных спорных ситуаций, касающихся распределения полномочий в проекте.

9.3 Производительность сотрудников. Знания и умения. Мотивация персонала

В силу того, что на поведение человека оказывают влияние многочисленные факторы, включая психологические и социальные, люди, работающие в проекте, не могут рассматриваться как обычные ресурсы. Люди склонны по-своему оценивать всю информацию, ставшую им доступной разными путями, преломлять ее через призму своего личного, уникального опыта и характера, и поступать, на первый взгляд, нерационально. Человеку чаще всего недостаточно слов "Вот задача, вот зарплата, – давай, работай!", чтобы начать делать то, что от него хотел бы руководитель проекта.

Перечислить все особенности управления персоналом достаточно трудно. Ниже приведен список лишь некоторых из них тех, с которыми весьма часто приходится сталкиваться на практике.

Производительность. Одной из наиболее специфических черт управления персоналом при разработке ПО являются особенности производительности людей. Разработка программ остается в большой степени творческой деятельностью, требует зачастую весьма специальных знаний и умений, глубокого понимания вопросов организации информации и аккуратного планирования работы, поэтому нельзя ожидать от людей, участвующих в ней, каких-то средних показателей производительности.

На производительность человека, занимающегося обдумыванием вопросов организации ПО, большое влияние оказывает окружение. Эта работа, с одной стороны, требует погружения, и разработчику необходимо достаточно много времени проводить в уединении, вдали от шума и суеты. С другой стороны, иногда требуется консультация эксперта, мнение руководителя или просто информация от других сотрудников по какому-то вопросу. Тишина, когда это нужно, и общение, когда оно необходимо, в правильном сочетании позволяют очень быстро продвигаться в решении трудных задач. А любая другая их комбинация – шум на рабочем месте и отсутствие возможности вовремя посоветоваться со специалистом – часто катастрофически сказывается на производительности такого труда. Определение индивидуальной производительности труда является одним из краеугольных камней управления персоналом в большинстве проектов.

Еще одна особенность разработки ПО связана с достаточно глубокой спецификой практически всех проектов и необходимостью обучения для адекватного вхождения в проект. Нельзя считать, что новый сотрудник сразу будет демонстрировать компетентность в вопросах, связанных с проектом. Обычно он несколько первых дней вносит в него отрицательный вклад, поскольку оттягивает на себя внимание и время других его участников, требует, чтобы его ввели в курс дела. Затем, постепенно, его вклад начинает расти и достигает максимума его возможностей за время от одного до шести месяцев, и зависимости от сложности проекта.

Соответственно, планирование хода проекта с надеждой на эту максимальную производительность с самой даты прихода разработчика и проект является самообманом. Замена одного из участников проекта на нового стоит значительно больше, чем разность их зарплат, а добавляя новых работников в проект, который отстает от графика, чаще всего вы только увеличите его отставание.

Другая сторона учета необходимости обучения связана с оценкой хода проекта. Часто дела в проекте через некоторое время после его начала идут медленнее, чем планировалось и хотелось бы. Это отставание от графика может быть вызвано влиянием периода обучения, во время которого сотрудники демонстрируют меньшую производительность. Опытные руководители в таких ситуациях не делают поспешных выводов и не торопятся признать проект провалившимся.

Знания и умения. Создание сложных программ предполагает умение работать с информацией и алгоритмами на достаточно высоком уровне абстракции, предвидеть свойства той или иной организации данных и действий над ними. Наблюдать эти навыки непосредственно практически невозможно, что-то сказать об их наличии у человека может лишь опыт участия в сложных проектах. При наборе людей в проект нужно помнить, что адекватную оценку разработчику ПО очень трудно дать только по результатам одного-двух разговоров с ним. Вместе с тем, часто приходится идти на риск, беря в проект новичков. Проект по разработке ПО не сможет стать успешным, если его команда состоит целиком из новичков, она всегда должна включать нескольких опытных и умелых разработчиков.

Мотивация персонала. Мотивация людей – один из самых сложных вопросов управления. Готовых рецептов в этом вопросе предложить нельзя, однозначно можно сказать лишь то, что мотивация сугубо индивидуальна и умение применять ее требует внимательного отношения к сотрудникам и понимания основных черт личности каждого из них. Считается, что потребности человека, определяющие основные цели его действий, образуют иерархию – *пирамиду Маслоу* (Maslow). Мотивация людей должна затрагивать разные их потребности и ни в коем случае не сводиться только к денежной составляющей, расположенной на втором-третьем уровне этой пирамиды. Социальные потребности побуждают людей посвящать часть своего времени семье и близким, общению с другими людьми, деятельности в рамках различных неформальных групп. Потребности в самореализации отвечает предоставление человеку определенной ответственности, возможности самому определять план и содержание своей работы, вести себя индивидуально и одежде, оформлении рабочего места, расположении вещей на столе (довольно часто организации пытаются регламентировать все эти вопросы).

9.4 Лидерство и влияние. Принципы построения сплоченной команды

Лидерство и влияние. Один из аспектов управления персоналом – развитие руководителем проекта своих собственных качеств как лидера, способного эффективно концентрировать усилия людей на нужных задачах, вести за собой и обучать персонал работе в условиях ограниченных ресурсов и высоких требований. Важная характеристика лидера – это *влияние,* которым он обладает. Влияние рассматривается как одна из составляющих власти, наряду с полномочиями, определяющими официальные рамки власти управленца в организации, и статусом, определяющим сложившееся отношение к занимаемой им позиции (а не к самому человеку, что как раз и является его личным влиянием).

Влияние руководителя поддерживается и укрепляется за счет нескольких факторов.

- Укрепление репутации эксперта в технических аспектах и предметной области, что повышает влияние на руководство организации, и реальных знаний в этой области и технологиях, что повышает доверие со стороны подчиненных.
- Акцент социальных взаимоотношений внутри организации, лежащий ближе к деловым аспектам, чем к предпочтениям и привычкам. Более важно поддерживать и укреплять связи с теми, кто может помочь в достижении деловых целей, а не с более статусными руководителями или старыми знакомыми. Нужно развивать связи с другими экспертами и специалистами, которые могут быть востребованы.

– Выбор правильной тактики общения в зависимости от его целей и другой стороны. Например, человеку труднее отказать в просьбе, если он обращается с ней в личной беседе, более легко – по телефону, еще легче – если просьба сформулирована в письме.

– Внимание к нуждам партнера, гибкость и способность идти на компромисс при четкой формулировке и твердой защите своих потребностей. Умение правильно истолковывать невербальные сигналы, жесты, уклонение от разговора и пр.

Большое значение для мотивации участников проекта и для успеха проекта в целом имеет **сплоченность его команды**. Сплоченные команды демонстрируют гораздо более высокую производительность, более эффективны при обучении новых служащих, их члены более заинтересованы в успехе проекта, более склонны к проявлению инициативы, менее подвержены разочарованиям от первых неудач, и т.д. Поэтому построение настоящей сплоченной команды является одной из важных задач руководителя проекта. Настоящую сплоченную команду нельзя сделать, сконструировать, – ее построение правильнее называть выращиванием. Как в педагогике или сельском хозяйстве, можно только посадить семена, создать благоприятные факторы и ждать, появится или нет нужный урожай.

Факторы, которые способствуют созданию сплоченной команды, таковы:

– *Разнородность начального состава и общие мотивации*. Людям легче сплотиться, если они обладают разными, дополняющими друг друга навыками и чертами характера, нуждаются в поддержке и помощи друг друга. При этом, конечно, нельзя перебарщивать и собирать в одну группу слишком разных людей с очень разными целевыми установками. Большую помощь в построении группы оказывают люди с внешней мотивацией, нацеленные на общение с коллегами.

– *Неформальные отношения и частое общение*. Сплочению способствует поддержка общения между членами группы, не обязательно относящегося к работе, но связанного с выполнением общей деятельности. Обычно в качестве стимулов к этому используют совместные поездки, совместные обеды и торжества, совместный отдых. Нужно поддерживать и общение на рабочие темы – совместные обсуждения целей проекта, основных работ, способов решения возникающих задач, проводить «мозговые штурмы». В частности, члены команды не должны сидеть в удаленных помещениях, тем более – в разных городах. Для обсуждения общих вопросов должно быть выделено специальное помещение.

– *Высокие стандарты качества*. Группа должна быть объединена общим стремлением сделать результаты проекта лучше. Это придает команде некоторый оттенок элитарности, который сближает ее членов.

– *Открытый стиль руководства*. Доверие к членам группы, поощрение инициативы, самостоятельности и самоорганизации, поощрение поведения, нацеленного на сотрудничество. Член сплоченной группы сам всегда готов оказать помощь коллеге и не постесняется попросить помощь у других. Он доверяет своим товарищам и знает, что они также полагаются на него.

– *Надлежащее техническое обеспечение работы команды и управление*. Организация удобной для работы среды. Сотрудники команды должны видеть, что руководитель действительно заботится о том, чтобы им было удобнее работать вместе, чтобы они могли работать над проектом по существу, а не терять время на бюрократические и организационные вопросы.

– *Поощрение индивидуальности и личностного отношения к работе, создание возможностей для самовыражения*. Используя все грани своей личности, человек может внести наибольший вклад в общую работу.

Есть и набор факторов, которые наоборот, препятствуют сплочению команды.

– *Неясные цели проекта, нудная и механическая работа*, постановка задач, которые не воспринимаются людьми как достойные и интересные.

– *Нехватка ресурсов и времени*, давление со стороны руководства, снижение требований к качеству результатов, сверхурочная работа. Команда обычно готова поработать немного больше, когда близкий к завершению проект чуть-чуть не попадает в нужный срок. Но использование свыше 2 часов сверхурочных работ в день и более недели в месяц очень плохо сказывается на качестве результатов, на работоспособности и на мотивации людей.

– *Защитный стиль руководства*, при котором руководитель пытается контролировать результаты всех подчиненных, не доверяет им полностью. Лишение подчиненных возможности самим принимать решения в рамках их работы и проявлять самостоятельность губит инициативу. Насаждение конкуренции между членами одной команды порождает недоверие и разрушает сплоченность.

– *Стандартизация и бюрократия*. Неудобная для работы обстановка – шумные, тесные офисы, труднодоступность мест для проведения обсуждений, географическое разделение команды, размещение в далеких друг от друга помещениях или даже в разных городах – делает совместную работу слишком трудной, чтобы она могла приносить удовольствие.

9.5 Управление конфликтами

При долгой совместной работе нескольких людей практически всегда неизбежны конфликты. Они не всегда имеют только отрицательные последствия, иногда конфликт возникает из-за различного понимания целей проекта или обуславливается проводимыми в организации изменениями. Некоторые такие конфликты иногда полезно искусственно подтолкнуть, иначе не получится выявить интересы вовлеченных в них лиц и содействовать более глубокому их удовлетворению при разрешении конфликта. Конфликты в проекте могут возникать по поводу его целей и приоритетов, по поводу административной политики и процедур, по поводу персонала и других ресурсов, времени и бюджета, из-за графика проекта, из-за технических разногласий по способам его реализации, а также из-за личных взаимоотношений людей. Руководителю, стремящемуся к успеху проекта, прежде всего нужно знать о возможных конфликтах по его поводу. Для этого надо выявить всех заинтересованных лиц и достаточно адекватно представлять их интересы, особенно интересы и потребности спонсоров, организации-исполнителя и организации-заказчика и их руководства. Нужно также научиться хорошо понимать людей, работающих в проектной группе. Выделяют пять основных методов поведения при конфликтах.

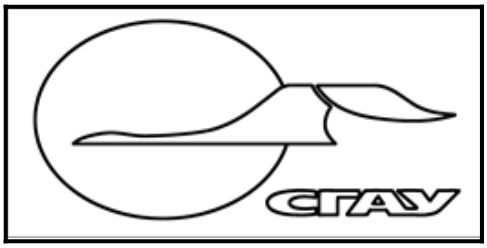
– *Уклонение.* Стремление избежать конфликтной ситуации, замалчивать и игнорировать в максимальной степени. Этот метод используется, если необходимо выиграть время для более полного осознания ситуации, когда проблема может разрешиться сама или когда возможный выигрыш, как и возможный проигрыш, достаточно малы. Уклонение не разрешает конфликт. Чаще всего, при этом конфликт только усиливается и может проявиться вновь в более серьезной форме.

- *Сглаживание.* Перенос внимания сторон на общие ценности, отказ от рассмотрения спорных вопросов при таком же поведении другой стороны. Такой метод применяется для сохранения хороших отношений, при необходимости избежать конфронтации для достижения общих целей. Он разрешает конфликт лишь на короткое время, которое может быть использовано сторонами для подготовки к более глубокому разрешению.
- *Силовое разрешение.* Принуждение одной из сторон принять точку зрения другой. Оно используется при необходимости быстро разрешить конфликт, когда одна из сторон более близка к позиции руководителя или находится в более выгодном положении, когда руководитель уверен в своих силах и не нуждается в одобрении своих действий. Этот метод разрешает конфликт на короткое или среднее время. Конфликт может проявиться позднее в другой форме.
- *Компромисс.* Этот метод требует от каждой из сторон пойти на некоторые уступки, что, однако, часто не дает им тех результатов, которых они хотели бы. Он используется для временного решения сложных проблем, для сохранения отношений между сторонами, не имеющими преимуществ друг перед другом, когда их возможные выигрыши не слишком высоки, а сохранение отношений имеет большое значение. Компромисс разрешает конфликт на среднее или долгое время.
- *Сотрудничество.* Открытое сопоставление точек зрения и интересов конфликтующих сторон с целью нахождения максимально выгодных им обеим решений. Применяется, если возможный выигрыш для обеих сторон достаточно велик, когда стороны питают взаимное уважение друг к другу и пытаются обеспечить долговременное сотрудничество, когда время, потраченное на разрешение конфликта, с лихвой окупится полученными преимуществами. Этот метод обеспечивает долговременное или полное разрешение конфликта.

Вопросы к экзамену

1. Общие понятия менеджмента разработки ПО. Проект и проектная деятельность.
2. Жизненный цикл проекта: этапы разработки ПО. Методология управления ИТ-проектами.
3. Структура работ, ресурсы ИТ-проекта. Фазы, процессы, итерации, вехи, роли, артефакты ИТ-решения.
4. Планирование проекта. Календарно-сетевое планирование. Методы планирования и управления проектами и ресурсами.
5. Среда MS Project. Планирование и контроль проектных работ.
6. Программные средства для управления ИТ-проектами: Open Plan, Spider Project, Primavera.
7. Средства управления версиями SVN. Средства Task Tracking и Bug Tracking.
8. Модели жизненного цикла ПО. Каскадная, итерационная, спиральная модели.
9. ГОСТ серии 34.
10. Capability Maturity Model for Software - модель зрелости процессов разработки ПО).
11. RUP.
12. MSF.
13. Personal Software Process / Team Software Process.
14. Оценка трудоемкости и сроков разработки программного обеспечения. Методика COCOMO II. Методология Crystal. Методология PRINCE.
15. Методология RAD – Rapid Application Development.
16. Гибкое управление разработкой. XP. Agile.
17. Scrum
18. Методология сервис-менеджмента: ITSM, ITIL
19. Стадии жизненного цикла ИТ-проекта.
20. Руководство PMBOK. Управление сроками.
21. Управление рисками: анализ, планирование, мониторинг и контроль.
22. Документация ИТ-проекта.
23. Организация проектной команды. RACI.
24. Производительность сотрудников. Знания и умения. Мотивация персонала. Лидерство и влияние.
25. Эскалация. Управление конфликтами.
26. Принципы построения сплоченной команды.

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)



СОГЛАСОВАНО	**УТВЕРЖДАЮ**
Управление образовательных программ	Проректор по учебной работе
_____/ А.В. Дорошин /	_____/ Ф.В. Гречников /
"_____"_____ 20___ г.	"____" _____ 20___ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Наименование модуля (дисциплины)

Менеджмент разработки ПО

Цикл, в рамках которого происходит освоение модуля (дисциплины)	Профессиональный цикл
Часть цикла	Вариативная часть
Код учебного плана	230100.68
Факультет	Информатики
Кафедра	Информационных систем и технологий

Курс	6		
Семестр	B		
Лекции (СЛ)	18		
Семинарские и практические занятия (СП)	0		
Лабораторные занятия (СЛР)	36	Экзамен	
Контроль самостоятельной работы / Индивидуальные занятия (КСР / ИЗ)	0	Зачет	A
Самостоятельная работа (СРС)	36		
Всего (Всего с экзаменами)	90		

Наименование стандарта, на основании которого составлена рабочая программа:

230100 Информатика и вычислительная техника Приказ № 554 от 09.11.09

Соответствие содержания рабочей программы, условий ее реализации, материально-технической и учебно-методической обеспеченности учебного процесса по дисциплине всем требованиям государственных стандартов подтверждаем.

Составители:

Иващенко А.В. _____
(подпись)

Заведующий кафедрой:

Прохоров С.А. _____
(подпись)

Рабочая программа обсуждена на заседании кафедры

Информационных систем и технологий

Протокол №___ от "____" _____ 20___ г.

Наличие основной литературы в фондах научно-технической библиотеки (НТБ) подтверждаем:

Директор НТБ _____ /_____/
(подпись) (расшифровка подписи)

Согласовано:

Декан _____ /_____/
(подпись) (расшифровка подписи)

1 Цели и задачи модуля (дисциплины), требования к уровню освоения содержания

1.1 Перечень развиваемых компетенций

использует на практике умения и навыки в организации исследовательских и проектных работ, в управлении коллективом (ОК-4);разрабатывать и реализовывать планы информатизации предприятий и их подразделений на основе Web- и CALS-технологий (ПК-3);применять современные технологии разработки программных комплексов с использованием CASE-средств, контролировать качество разрабатываемых программных продуктов (ПК-6);организовывать работу и руководить коллективами разработчиков аппаратных и/или программных средств информационных и автоматизированных систем (ПК-7)

1.2 Цели и задачи изучения модуля (дисциплины)

Целями освоения дисциплины являются формирование у будущих специалистов теоретических знаний и практических навыков по применению методик управления проектами в области информационных технологий, а также формирование профессиональных компетенций в части

выполнения проектных работ по автоматизации и информатизации прикладных процессов и управлению проектами информационных технологий (ИТ-проектами) по созданию и эксплуатации информационных систем (ИС).В курсе изучаются модели процесса разработки программного продукта, такие как SW-CMM, RUP, Microsoft Solutions Framework, Agile Development Methods. Также в курсе освещаются международные рекомендации по управлению проектами (PMBOK, Project Management

Body of Knowledge) Института управления проектами (Project Management Institute).Основной задачей курса является стартовая подготовка студентов в области организации разработки сложных программных комплексов, ознакомление их с современны-ми подходами и международным опытом в этой области

1.3 Требования к уровню подготовки студента, завершившего изучение данного модуля (дисциплины)

В результате освоения дисциплины студент должензнать основные технологии менеджмента проектов по разработке ПО, уметь применять применять их на практике при руководстве и участии в разработке ПО для мобильных устройств и уметь организовывать работу и руководить коллективами разработчиков ПО

1.4 Связь с предшествующими модулями (дисциплинами)

Методы проектирования и поддержки требований к программному обеспечениюАрхитектура современных распределенных системТехнологии и инструментальные средства разработки ПО для мобильных устройств

1.5 Связь с последующими модулями (дисциплинами)

Научно-исследовательская работа магистра

2 Содержание рабочей программы (модуля)

Семестр 1		
СЛ 0,2 18 часов 0,5 ЗЕТ	Активные 0	Общие понятия менеджмента разработки ПО. Проект и проектная деятельность. Жизненный цикл проекта: этапы разработки ПО. Методология управления ИТ-проектами. Структура работ, ресурсы ИТ-проекта. Фазы, процессы, итерации, вехи, роли, артефакты ИТ-решения.
		Планирование проекта. Календарно-сетевое планирование. Методы планирования и управления проектами и ресурсами. Среда MS Project.
		Планирование и контроль проектных работ. Программные средства для управления ИТ-проектами: Open Plan, Spider Project, Primavera. Средства управления версиями SVN. Средства Task Tracking и Bug Tracking.
		Модели жизненного цикла ПО. Каскадная, итерационная, спиральная модели. ГОСТ серии 34. Capability Maturity Model for Software - модель зрелости процессов разработки ПО). RUP. MSF. Personal Software Process / Team Software Process.
		Оценка трудоемкости и сроков разработки программного обеспечения. Методика COCOMO II. Методология Crystal. Методология PRINCE. Методология RAD – Rapid Application Development.
		Гибкое управление разработкой. XP. Agile. Scrum
		Методология сервис-менеджмента: ITSM, ITIL
		Стадии жизненного цикла ИТ-проекта. Руководство PMBOK. Управление сроками. Управление рисками: анализ, планирование, мониторинг и контроль. Документация ИТ-проекта.

		Организация проектной команды. RACI. Производительность сотрудников. Знания и умения. Мотивация персонала. Лидерство и влияние. Эскалация. Управление конфликтами. Принципы построения сплоченной команды.
	Интерактивные 0	
	Традиционные 0	
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СЛР 0,4 36 часов 1 ЗЕТ	Активные 0	Календарно-сетевое планирование. Расчет критического пути
		Составление сетевого графика в MS Project. Планирование ресурсов и затрат.
		Анализ и оптимизация проекта в MS Project. Анализ отклонений. Управление сроками.
		Расчет экономической эффективности ИТ проекта
		Организация работы в SVN. Контроль версий.
		Изучение возможностей систем Task tracking и Bug tracking.
		Планирование проекта в SCRUM
		Оценка рисков проекта заданной предметной области
		Оценка экономической эффективности проекта по разработке ПО
	Интерактивные 0	
	Традиционные 0	
КСР 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СРС 0,4 36 часов 1 ЗЕТ	Активные 0	Достоинства и недостатки различных моделей жизненного цикла ПО
		ГОСТ 34.201-89, ГОСТ 34.602-89. Руководящий документ РД 50-34.698-90.

		Руководство PMBOK. Сертификация PMP
		Планирование проектов. Методы. PERT
		Достоинства и недостатки гибких технологий управления проектами
		CASE средства Scrum
		Методика Total Cost Ownership (TCO). Методика Rapid Economic Justification (REJ)
		Программные средства для управления проектами: Open Plan, Spider Project, Primavera.
		Современные технологии тестирования и контроля качества ПО
	Интерактивные 0	
	Традиционные 0	

3 Инновационные методы обучения

Проведение лабораторных работ в игровой форме (Scrum, планирование проектов, управление рисками, совместная разработка ПО)

4 Технические средства и материальное обеспечение учебного процесса

Компьютеры, оргтехника, проектор;

5 Учебно-методическое обеспечение

5.1 Основная литература

1 Богданов, В.В. Управление проектами в Microsoft Project 2007 [Текст] : учеб. курс / Вадим Богданов. - СПб. [и др.] : Питер : Питер Пресс, 2007. - 592 с. + 1 эл. опт. диск (CD-ROM).2 Уткин, В.Б. Информационные системы в экономике [Текст] : [учеб. по специальности "Прикладная информатика" (по обл.) и др. междисциплинар. специальностям] / В. Б. Уткин, К. В. Балдин. - 3-е изд., стер. - М. : Академия, 2006. - 283 с.3 Балдин, К.В. Информационные системы в экономике [Текст] : учебник : [для вузов по специальности 351400 "Прикладная информатика (по обл.)" и др. междисциплинар. специальностям] / К. В. Балдин, В. Б. Уткин. - 5-е изд. - М. : Дашков и К, 2007. - 394 с.

5.2 Дополнительная литература

1 Козырев, А.А. Информационные технологии в экономике и управлении [Текст] : учебник / А. А. Козырев. - Изд. 4-е. - СПб. : Изд-во Михайлова В. А., 2005. - 444 с. 2 Разработка программных проектов на основе Rational Process (RUP) [Текст] / Гари Поллис, Лиз Огастин, Крис Лоу, Джас Мадхар ; пер. с англ. под ред. А. П. Караваева. - М. : Бином : Бином-пресс, 2005. - 255 с.3 Грабауров, В.А. Информационные технологии для менеджеров [Текст] / В. А. Грабауров. - М. : Финансы и статистика, 2002. - 367 с.4 Грей, Клиффорд Ф. Управление проектами [Текст] : практ. рук. : пер. с англ. / Клиффорд Ф. Грей, Эрик У. Ларсон. - М. : Дело и сервис, 2003. - 527 с.5 Управление проектами [Текст] : [пер. с англ. / П. У. Г. Моррис, Д. И. Клилэнд, Р. А. Лундин [и др.]; под ред. Дж. К. Пинто. - СПб. [и др.] : Питер : Питер принт, 2004. - 463 с.6 О'Коннэл, Фергус. Как успешно руководить проектами. Серебряная пуля [Текст] : пер. с англ. / Фергус О'Коннэл . - М. : КУДИЦ-ОБРАЗ, 2004. - 287 с.

5.3 Электронные источники и интернет ресурсы

5.4 Методические указания и рекомендации