ШАГ
компьютерная
АКАДЕМИЯ

УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ

УРОК №1

ВВЕДЕНИЕ В УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

Причины возникновения дисциплины управление программными проектами

Анализ разработки ПО показал, что огромное количество проектов разрабатывается с отклонениями от технического задания на проект, сроков реализации проекта и всегда выходит за рамки бюджета. Причин много. Назовем некоторые вместе со следствиями.

1. **Заказчик не представляет возможностей разработки и применения**

Следствие: в процессе разработки у заказчика появляются совершенно новые взгляды на то, что он хотел бы получить.

2. **Заказчик не понимает сложности разработки.**

Следствие: бюджет и сроки выполнения устанавливаются совершенно невыполнимыми.

3. **Исполнитель не знает предметной области и не в состоянии оценить сложность задачи.**

Следствие: затягивание сроков, непомерное раздувание бюджета, нарушение бюджета и сроков, неверное решение или его отсутствие вовсе.

4. **Изменение существующего положения вещей,** начиная от возникновения новых методов и технологий разработки или замены технологий у заказчика и заканчивая крахом фирмы заказчика или исполнителя.

Возможны различные комбинации этих и других причин, что приводит к огромному многообразию объяснения краха большинства проектов (по оценкам – до 30% провалившихся и всего около 30% завершенных). Это привело к необходимости страховки как заказчика, так и исполнителя от неуспешного завершения, которую предоставляет компьютерная инженерия. Одной из составных частей компьютерной инженерии является дисциплина **«Управление проектами»**. В управлении проектами речь идет об организации процесса разработки ПО.

Процесс разработки ПО можно разбить на несколько составных частей, называемых **жизненным циклом проекта**.

Стадии жизненного цикла:

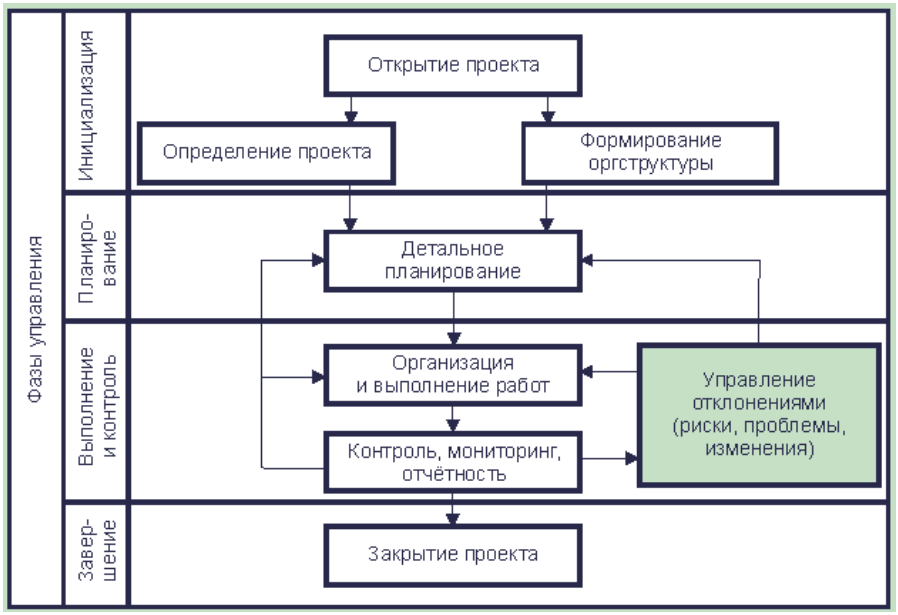

Рисунок 1

На рисунке 1 показано, что стадии жизненного цикла программы делятся на неусловные фазы, каждая из которых имеет свое назначение. Так, фаза открытия проекта, которая часто называется фаза намерений сторон. Фаза планирования раньше называлась разработкой технического задания, которое было незыблемым до окончания проекта. Рисунок показывает, что сейчас предусмотрено влияние фазы выполнения на фазу планирования, которая в таком случае уже называется перепланированием.

Диаграммы Ганта

Процесс разработки можно разбить на составные части, для каждой из них определить длительность и последовательность выполнения. При этом рассматриваются действия, последовательность которых неизменна относительно друг друга. Например, нельзя разработать форму для ввода свойств объекта, если мы не определили, какими свойствами обладает этот объект. Или пример попроще: нельзя отладить часть программы, которая еще не написана. Для отображения последовательных составных частей какого-либо процесса удобно воспользоваться **диаграммами Ганта** (*исследователь процессов производства начала XX столетия, первая диаграмма была предложена в 1910 году – более 100 лет назад!*) – горизонтальные столбчатые диаграммы, каждый новый вид работ отображается на новой строке.

Например, напишем план выполнения домашнего задания:

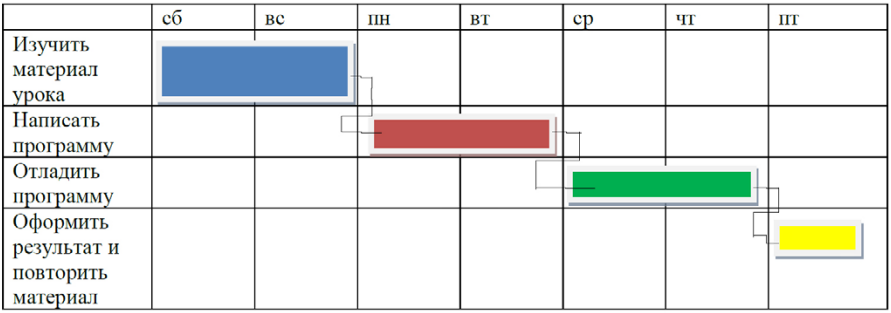

Рисунок 2

Диаграммы Ганта используются для составления планов. Каждый столбик соответствует отдельному действию. Действия, составляющие план, размещаются по вертикали. Начало, конец и длина столбца на шкале времени соответствуют началу, концу и длительности работы. Можно указывать зависимость между работами. Диаграмма может использоваться для представления текущего состояния выполнения работ: часть прямоугольника, соответствующая выполненной части работы, заштриховывается. Вертикальной линией отмечают текущий момент. К диаграмме прикладывается таблица со списком работ, строки которой соответствуют отдельно действию из диаграммы, а столбцы содержат дополнительную информацию о планируемой работе.

Что такое проект и программный проект

Программное обеспечение (ПО) – это программа или группа программ, которая является конечным продуктом проекта программной разработки. Программную разработку называют программным инжинирингом.

Институт программного инжиниринга (Software Engineering Institute, SEI) дал следующее определение программного обеспечения:

> «***ПО*** *– программы, процедуры и символические языки, которые управляют функционированием аппаратных средств».*

Проект – это спланированная заранее последовательность действий.

Институт управления проектами (Project Management Institute, PMI) дал следующее определение проекту:

> *«**Проект** это:*
>
> - *Определенный план или разработка проекта: Схема;*
> - *Запланированное действие:*
>
> *(а) определенным образом сформулированная часть исследования;*
>
> *(б) большое, обычно поддержанное субсидиями некоторых организаций или правительства действие;*
>
> *(в) задача или проблема, обычно поставленная перед группой студентов, которая затем выносится на аудиторные занятия».*

Еще одно определение касается управления разработкой. Это определение менеджмента:

> ***Менеджмент** – это практика выполнения проекта и управления проектом.*

Определение инжиниринга ПО

Согласно Барри Боэму *(Barry Boehm):*

*«**Инжиниринг ПО** – это практическое применение научных знаний при разработке и создании компьютерных программ и связанной с ними документации, необходимой для их разработки, использования и поддержки».*

(Boehm B. @Software Engineering Economics», Englewood Cliffs, NJ: Prentice Hall, 1976.– p.16).

Институт IEEE дает такое определение инжиниринга:

*«**Инжиниринг ПО** – систематический подход к развитию, действию, поддержке и прекращению эксплуатации ПО».*

(Institute of Electrical and Electronics Engineers. IEEE Std 610.12.1990 Standard Glossary of Software Engineering Terminology. «Software Engineering Collection». NY: Institute of Electrical and Electronics Engineers, 1983.– p.76).

Согласно Стефану Шаху *(Stephen Schach):*

*«**Программный инжиниринг** – дисциплина, целью которой является создание качественного ПО, которое завершается вовремя, не ведет к превышению выделенных бюджетных средств, а также удовлетворяет выдвигаемым требованиям».*

(Schach Stephen R. «Classical and Object-Oriented Software Engineering» 4th ed. Boston, MA: McGrawHill, 1999.– p.4).

Если объединить эти определения, то получится определение, наиболее полно отражающее суть:

***Инжиниринг программного обеспечения** – это регламентированная, системная методология разработки, использования, обслуживания и прекращения эксплуатации ПО на основе практического применения научных знаний.*

Определение проекта

Определение, данное Джеймсом Льюисом *(James Lewis):*

*«**Проект** – одноразовая работа, которая имеет определенные даты начала и окончания, ясно определенные цели, возможности и, как правило, бюджет. Это действия, отличающиеся от повторяющихся операций, таких как производство, обработка заказа и т.д. В данном случае идет речь о специфическом действии, имеющим определенные тактические цели».*

(Lewis James P. «Project Planning, Schedulling and Control: A Hands-On Guide to Bringing Projects in on Time and on Budget», rev ed. Chicago, IL: Irwin, 1995.– pp. 2–3).

По Гарольду Керцнеру *(Harold Kerzner)* ***проект это*** *–*

«Произвольный ряд действий или задач, имеющий определенную цель, которая будет достигнута в рамках выполнения некоторых заданий, характеризующихся определенными датами начала и окончания, пределами финансирования (в случае прикладного проекта) и ресурсами (деньги, трудозатраты, оборудование)».

(Kerzner H. «Project Management: A System Approach to Planning, Schedulling and Controlling» 6th ed. NY: John Wiley & Sons, 1998 – p.2).

Эти определения собираются в результирующий треугольник «Время-Качество-Деньги» (в простонародье – «Быстро-Качественно-Дешево» – по этой теме есть хорошо известная шутка, суть которой сводится к следующему: «Оставьте только два пункта» ☺). В процессе выполнения проекта решается задача сохранить стоимость в заданных пределах, выдержать сроки выполнения при некотором определенном уровне качества. Задача управления проектом как раз и состоит в том, чтобы

уравновесить эти три составляющие: стоимость, график выполнения проекта и качественный результат. Но так не происходит обычно. График, бюджет и качество не выдерживаются на требуемом уровне. Поэтому руководители проекта вынуждены выбирать в виде конечной цели только один или два параметра треугольника. Как уже говорилось выше – на профессиональном сленге это действие известно как «Качественно-быстро-дешево – выбери два из них».

Институт управления проектами (PMI) дает такое определение проекта:

> «***Проект*** *это временное усилие, предпринятое для того, чтобы создать уникальный продукт или услугу с определенной датой начала и окончания действия, отличающегося от продолжающихся, повторных действий и требующего прогрессивного совершенствования характеристик».*

Эти определения проекта имеют общие черты:

- **Цель.**

Должна быть четко определенна цель или ряд целей проекта. По завершению проекта должен быть получен какой-либо результат. Если проект предусматривает достижение нескольких целей, то они должны быть взаимосвязаны и непротиворечивы.

- **Момент начала и завершения проекта.**

Проект имеет протяженность во времени. У него есть четко определенное начало и конец действия, связанное с временной шкалой каких-либо дат. Поддержка ПО не является проектом и обычно представляет собой пролонгированное за пределы проекта действие и, но

может быть включена в проект (например, в качестве отдельных версий).

- **Уникальность.**

Проект – одноразовая сущность, не повторяющаяся при повторении такого же по сути проекта, но повторяющаяся работа тоже может являться проектом. Постройка дома обычно определяется как проект несмотря на то, что подрядчики сконструировали уже десятки зданий. Пусть образец и процесс в основном совпадают, выполняются по шаблону, но имеется достаточно различий в каждом доме (это коллектив, размещение, материалы и т. п.). Иначе речь идет о поточной линии, где идентичные части выполняются аналогичным образом. То же самое справедливо для профессионалов в области разработки ПО – они никогда не создают идентичную программную систему, хотя могут ее копировать и составлять из имеющихся решений переносить произвольным образом.

- **Ограничения.**

В проекте есть ограничения по стоимости, графику разработки и качеству выполнцения. Эти ограничения формируют треугольник, который для достижения успеха должен быть сбалансирован и управляем.

(Project Management Institute. «A Guide to the Project Management Body of Knowledge» Sylva, NC: PMI Publication Division. 1996.– p.167).

Итак, проект, в терминах разработки ПО:

Проект *– это уникальное, временное действие с определенными датами начала и конца, направленное на то, чтобы достичь одной или нескольких целей при ограничениях по стоимости, графику и качеству выполнения.*

Что такое управление проектами?

Институт управления проектами (PMI) определяет ***управление проектами*** как:

> *«набор проверенных принципов, методов и методик, применяемых для эффективного планирования, составления графика, управления и отслеживания результатов работы, ориентированной на успешное выполнение, а также определяющих базис для планирования проектов».*

Керцнер дает определение ***управлению проектами*** как:

> *«планирование, организацию, контроль и управление ресурсами компании, выделенными в рамках определенного проекта. Эти ресурсы предназначаются для достижения краткосрочной цели, которая была установлена для удовлетворения определенных целей и намерений».*

Отметим общие мысли в этих определениях:

- **управление** – умения в области управления проектами – это часть общих умений в области управления;
- **навыки** – навыки в области управления проектами – это использование умений в области управления для достижения запланированного результата. Они включают и планирование, и организацию производственного процесса, и составление графика, и контроль, и управление, и анализ.

Итак, резюмируя, получаем определение управления программными проектами.

> ***Управление программными проектами*** *– специализация общего менеджмента, которая определяет применение*

руководящих стандартных навыков планирования, комплектования персоналом, организации а также управления и контроля для достижения заранее определенной цели проекта.

Что такое командная разработка?

Были времена, когда программист самостоятельно определял, проектировал, писал и проверял свою работу. Этому способствовало окружение: простая однопользовательская природа используемых средств, небольшие размеры разрабатываемых приложений. Это позволяло считать разработку ПО индивидуальной деятельностью. Но в некоторый момент времена изменились: «Разработка программного обеспечения стала командным видом спорта» *(Г. Буч, 1998).*

Что же такое командная разработка, команда, группа? Группа может состоять из любого числа людей, взаимодействующих друг с другом, психологически принимают других членов группы и рассматривают себя, как часть группы.

Команда – это группа, состоящая из членов, которые влияют друг на друга ради достижения общей цели. Главное отличие команды от группы,– это результат эффективного взаимодействия между людьми на основе общих устремлений и ценностей, а также взаимодополняющих умений (*skills*), что приводит к тому, что суммарное усилие команды намного превышает сумму усилий ее отдельных членов. Как и в спорте, командная работа чрезвычайно важна для решения проблем конкурентоспособности

на общем рынке, где индивидуальное мастерство менее важно, чем высокий уровень коллективных действий. В современном мире информационных технологий команда – это единственное условие выживания, правило, а не исключение. Характерной чертой команды является широкий круг полномочий или прав принятия решения каждого члена команды.

Команда разработчиков программного обеспечения должна обладать следующими профессиональными навыками:

- умение правильно понять проблему, решение которой призвано решить разрабатываемое программное обеспечение (ПО);
- способность выявления требований, предъявляемых к разрабатываемому ПО посредством общения с пользователем системы и другими заинытересованными лицами;
- умение преобразовать понимание проблемы и потребностей клиентов в исходное определение системы, которое будет удовлетворять эти потребности;
- умение управлять масштабом проекта;
- умение уточнять определение системы до уровня детализации, пригодного для проектирования и реализации;
- способность оценить правильность разрабатываемого ПО, проведения его верификации и управление изменениями.

Командная разработка – это процесс разработки программного обеспечения, который реализует:

- руководство деятельностью команды;
- управление задачами отдельного работника и команды в целом;
- указания, какие компоненты следует реализовывать;
- предоставляет критерии для отслеживания и измерения продуктов и функционирования проекта.

Важным фактом в командной разработке является то, что члены команды имеют различия в профессиональных навыках и умениях. Баланс и разнообразие навыков – это две наиболее важные составляющие отличной команды. Именно это и делает команду командой. Одни «игроки» способны эффективно работать с заказчиками, другие имеют навыки программирования, третьи умеют тестировать программы, четвертые – это специалисты по проектированию и архитектуре систем.

Анализ проблем одиночной и командной разработки программного обеспечения

В настоящее время успешно используются и одиночное и командное изготовление ПО. Но сегодня абсолютное большинство крупных проектов – результат коллективного труда. В то же время одиночки – это и многочисленные фрилансеры и, программисты, совмещающие работу на компанию с одиночной разработкой.

Инженер компании Sun Microsystems Итан Николс, испытывая крайнюю стесненность в средствах к суще-

ствованию (недавно стал отцом), вынужден был искать источники дохода и вдохновился примером успеха независимого разработчика iPhone-приложений Стива Деметера *(Steve Demeter).* Итан Николс разработал головоломку Trism, которая принесла ему около 600 000 долларов дохода (http://www.iphones.ru/iNotes/17470). Другой пример программиста-одиночки – это создатель «БитТоррент», американский программист Брэм Коэн.

Давайте попробуем проанализировать особенности одиночной и командной разработки ПО:

Одиночная разработка ПО	Командная разработка ПО
Необходимость самому находить заказчиков и вести с ними переговоры (возможно на английском языке) об объеме, качестве, сроке и стоимости работ	Поиском заказчиков и сопровождением разработки профессионально занимаются специально обученные сотрудники компании
Принятие всех решений (язык программирования, архитектура, дизайн приложения и др.) осуществляется единолично на основе предыдущего опыта конкретного разработчика	Принятие всех решений (язык программирования, архитектура, дизайн приложения и др.) осуществляется с учетом мнения нескольких разработчиков основе предыдущего опыта
Заказчик напрямую общается с разработчиком, что исключает эффект «испорченного телефона»	Заказчик не всегда напрямую общается с разработчиком, что может внести эффект «испорченного телефона»
Разработчик свободен в выборе заказчика, времени и места работы	Разработчик ограничен работой в проектах компании согласно штатного расписания компании
Разработчик более подвержен рискам неоплаты (неполной оплаты) выполненной работы	Доход разработчика за выполненную работу, как правило, гарантирован компанией

Анализ терминов предметной области

Мы проанализируем следующие термины: **процесс**, **проект**, **персонал**, **продукт**, **качество**.

То, как мы создаем ПО, какую последовательность действий при этом выполняем, как взаимодействуем с другими членами команды, каких норм и правил придерживаемся в работе, как наш проект взаимодействует с внешним миром называется процессом разработки ПО. Основой успешности разработки ПО является методологически правильное выстраивание процесса разработки, его осознание и улучшение.

Процесс создания ПО представляет собой множество различных видов деятельности, методов, методик, таких как разработка технического задания, архитектуры приложения, кодирование тестирование, написание документации и др.

На сегодняшний день не существует универсального процесса разработки ПО, следуя которому мы можем гарантированно получить качественный продукт в оговоренные сроки. Каждая конкретная разработка, которая осуществляется некоторой командой разработчиков имеет большое количество индивидуальных особенностей. Но все же перед началом проекта необходимо спланировать процесс работы, определив роли и обязанности в команде, планы и сроки выполнения промежуточных и окончательной версий продукта.

Результатом **процесса разработки** программного обеспечения и итогом проекта является **продукт,** готовый к поставке (**конечный продукт**). Он включает в себя тело – исходный код, в виде компонентов, которые могут

быть откомпилированы и выполнены, руководство по эксплуатации и дополнительные составляющие части поставки. Прежде чем продукт становится готовым к поставке, он проходит стадию рабочего продукта.

Рабочий продукт – это промежуточные результаты процесса разработки ПО, которые помогают идентифицировать, планировать и оценивать различные части результата. Промежуточные результаты помогают менеджерам разных уровней отслеживать процесс реализации проекта, а заказчик получает возможность ознакомиться с результатами задолго до окончания проекта. Разработчики проекта в своей ежедневной работе получают простой и эффективный способ обмена рабочей информацией – обмен результатами.

Разработка ПО относится к проектным видам деятельности. Проекты подразделяются на промышленные и творческие, соответственно с разными принципами управления. Так вот, разработку ПО относят к творческим проектам.

Промышленные проекты часто объединяют большое количество разных организаций при невысокой уникальности самих работ, например – строительство многоэтажного дома. К ним можно отнести различные международные проекты и не только промышленные – образовательные, культурные и пр. Главная задача в управлении такими проектами – это все предусмотреть, всех проконтролировать, ничего не забыть, все суметь собрать «до кучи», добиться согласованного взаимодействия.

Творческие проекты характеризуются абсолютной новизной идеи – новый вид обслуживания, совершенно

новый программный продукт, не имеющий аналогов на рынке, сюда также можно отнести проекты в области искусства и науки. Всякий начинающий бизнес, обычно, становится таким вот творческим проектом. Новизна в подобных проектах не абсолютная – такое может, уже и было, но для команды проекта – в первый раз. Тут говорится об огромном объеме новизны для самих людей, которые воплощают этот проект. Проекты по разработке программного обеспечения впитывают части промышленных проектов и, несомненно, проектов творческих, находясь между этими полюсами. Часто они сложны потому, что объемны и находятся на стыке различных дисциплин – того целевого бизнеса, где должен применяться программный продукт, и сложного, нетривиального программирования. Часто в программные проекты добавляется разработка уникального оборудования. С другой стороны, так как программирование активно проникает в разные сферы человеческой деятельности, то происходит создание абсолютно новых, уникальных программных продуктов, и их разработка и продвижение обладают всеми чертами творческих проектов.

На этапе разработки, производятся регулярные измерения разработанного продукта, для определения соответствия требованиям. Любые несоответствия должны рассматриваются как дефекты – отсутствие качества.

Качество продукта определяется в стандарте ISO 9000:2005 как степень соответствия его характеристик требованиям. Качественного программного продукта не создать без качественного процесса разработки.

Методами обеспечения качества ПО являются:

1. Создание и совершенствование качественного процесса разработки ПО.
2. Обеспечение качества кода, путем соблюдения стандартов оформления кода в проекте, и контроль за соблюдением этих стандартов. Сюда входят правила на создание идентификаторов переменных, методов и имен классов, на оформление комментариев и т. д. Качество кода обеспечивается также рефакторингом – переписыванием кода, но не с целью добавления новой функциональности, а для улучшения его структуры.
3. Тестирование – самый распространенный на сегодня метод без которого не выпускается сегодня ни один продукт.

Если ПО создается командными усилиями, то высокое качество и продуктивность появляются, когда **персонал,** т. е. члены команды эффективно вовлечены в общее дело и сохраняют мотивацию, а вся команда действует эффективно. Для управления командой требуется перейти в сферу отношений между людьми. Как правило, используется иерархическая структура команды. Возглавляет команду менеджер проекта, который подчиняется лидеру направления (начальнику отдела). Обычная команда состоит из разработчиков, администратора баз данных, тестеров ПО и возможно других специалистов. Крупный проект может включать лидеров модулей, каждый из которых имеет в своем подчинении нескольких разработчиков.

Цель менеджера проекта заключается в том, чтобы получить уверенную в своих силах команду. При этом необходимо учитывать следующие человеческие факторы:

- квалификацию, подготовку и опыт членов команды;
- личные стремления и карьерный рост членов команды;
- потребности в наставничестве и развитии.

Очень важным моментом является общение в команде. Команда, члены которой будут совместно работать несколько месяцев ради достижения общей цели, должна составлять единое целое, между ее членами должно быть полное взаимопонимание.

Для поддержки информированности членов команды о продвижении проекта и его проблемах менеджеры проектов могут пользоваться следующими методами:

- организация досок объявлений для сообщений, отчетов о проекте;
- электронные рассылки по проекту;
- internet (intranet)-сайт для публикации документов, связанных с проектом;
- совещания по проекту.

Подводя итоги, дадим определения терминам.

Проект *– организационная сущность, которая используется для управление разработкой ПО. В результате проекта появляется программный продукт.*

Персонал *– разработчики, архитекторы, тестеры, руководители, пользователи, заказчики и все остальные заинтересованные лица – это основная движущая сила программного проекта.*

Качество *– характеристика ПО как степени его соответствия требованиям. Соответствие требованиям предполагает, что требования должны быть настолько четко определены, что они не могут быть поняты и интерпретированы некорректно.*

Продукт *– артефакты (от латинского artefactum – нечто искусственно сделанное), создаваемые в процессе деятельности проекта. К ним относятся модели, тексты программ, исполняемые файлы и документация.*

Процесс создания ПО *– это определение полного набора видов деятельности, необходимых для преобразования требований пользователя в продукт.*

Характеристики проекта

Из характеристик проекта можно выделить следующие:

- **тип проекта**;
- **цель проекта**;
- **требования к качеству;**
- **требования к бюджету;**
- **требования по срокам завершения.**

У проекта по разработке ПО присутствует четыре характерных признака:

1. Направленность на достижение конкретных целей.
2. Координированное выполнение взаимосвязанных действий.

3. Ограниченная протяженность во времени с определенным началом и концом и ограничения в используемых ресурсах (качественно – быстро – дешево).
4. Неповторимость и уникальность (хотя бы частичная).

Это и есть признаки, отличающие проекты от других видов человеческой деятельности.

Проекты направлены на достижение конкретных целей

Все усилия по планированию и реализации проекта предпринимаются для того, чтобы его цели были достигнуты, получены конкретные результаты. Именно цели являются движущей силой проекта.

Поэтому важно при управлении проектом точное определить и сформулировать цели, начиная с верхнего уровня и до мельчайшей детализации целей и задач.

Координированное выполнение взаимосвязанных действий

Проекты всегда имеют составные части и персонал, реализующий определенные задачи. Некоторые части проекта являются зависимыми, действия персонала не бывают изолированными. Для их правильного обьединения и получения итогового результата нужно организовать их взаимодействие. Нарушение синхронизации выполнения разных частей ставит под угрозу выполнение проекта. Проект – это динамическая система, складывающаяся из взаимосвязанных частей и требующая особых подходов к управлению.

Ограниченность средств

Денег всегда недостаточно. Времени всегда мало. Кроме того, проект ограничен рамками законов государства. Необходимо помнить, о персонале, который имеет какие-то моральные принципы, что тоже может быть своего рода ограничением. И, в конце концов, проект ограничен законами природы. Вот основные ограничения проекта.

Неповторимость и уникальность

Из-за постоянного изменения условий реализации проекта планирование должно быть гибким до такой степени, что могут меняться даже первоначальные цели и методы их достижения. Невозможно предсказать заранее все изменения, которые будут сопровождать выполнение проекта.

Закон Лермана гласит: «*Любую техническую проблему можно преодолеть, имея достаточно времени и денег*».

У закона есть следствие *(Лерман): «Вам никогда не будет хватать либо времени, либо денег»***.** Методика управления деятельностью на основе проекта была разработана именно для преодоления сформулированной в следствии Лермана проблемы. Обычно руководитель проекта понимает свою основную задачу в выполнении проекта как обеспечение выполнения работ.

Более опытный руководитель проекта точнее сформулирует определение главной задачи менеджера проекта: «*Обеспечить выполнение работ в срок, в рамках выделенных средств и в соответствии с техническим заданием*». Именно эти три составляющих: время, бюджет и качество работ находятся под постоянным вниманием

руководителя проекта. Поэтому еще расширим определение проекта:

> ***Проектом*** *называется деятельность, направленная на достижение определенных целей с максимально возможной эффективностью при заданных ограничениях по времени, денежным средствам и качеству конечного продукта.*

Для того чтобы справиться с ограничениями по времени, используются методы построения и контроля графиков работ. Для управления денежными ограничениями используются методы формирования финансового плана (бюджета) проекта и, по мере выполнения работ, соблюдение бюджета отслеживается, с тем чтобы не дать затратам выйти из-под контроля.

Пример:

№ этапа работ	Описание этапа работ	Сложность этапа работ	Трудоемкость этапа работ (на основании данных из предыдущих проектов), план	Трудоемкость этапа работ, факт	Стоимость этапа работ, план	Стоимость этапа работ, факт
1	Обновление персональных данных	высокая	8 человеко-часов			
2	Добавление адреса	средняя	6 человеко-часов			
3	Обновление адреса	средняя	6 человеко-часов			
4	Удаление адреса	низкая	4 человеко-часа			

№ этапа работ	Описание этапа работ	Сложность этапа работ	Трудоемкость этапа работ (на основании данных из предыдущих проектов), план	Трудоемкость этапа работ, факт	Стоимость этапа работ, план	Стоимость этапа работ, факт
5	Добавление телефонного номера	средняя	6 человеко-часов			
	...					
	ИТОГО:					

Разрабатывать программное обеспечение очень непросто, а разрабатывать качественное программное обеспечение и при этом завершать работу в срок – еще сложнее.

Согласно Jet Info Online в конце 60-х годов прошлого века в США произошел кризис ПО *(software crisis)*. Сюда входило отставание от графика, превышение сметы в больших проектах.

Кроме того, разработанное ПО не обладало контрактными функциональными возможностями, имело низкую производительность и его качество не удовлетворяло потребителей.

По результатам исследований, произведенных в 1995 году компанией Standish Group, проанализировавшей работу 364-х американских компаний, результаты выполнения более 23-х тысяч проектов, связанных с разработкой ПО, выглядели следующим образом:

- только 16,2% завершились вовремя, реализовали все требуемые функции и возможности и не превысили запланированный бюджет;

- 52,7% проектов завершились с опозданием, требуемые функции не были реализованы в полном объеме, расходы превысили запланированный бюджет;
- 31,1% проектов были прекращены до завершения;
- для двух последних категорий проектов срок выполнения среднего проекта оказался превышенным на 122%, а бюджет – на 89%.

В 1998 году процентное соотношение трех перечисленных категорий проектов лишь немного изменилось в лучшую сторону (26%, 46% и 28%, соответственно).

По оценкам ведущих аналитиков, незначительное изменение в лучшую сторону процентного соотношения трех перечисленных категорий проектов в последние годы происходит, в основном, не за счет повышения управляемости и качества проектирования, а за счет снижения масштаба выполняемых проектов.

В числе причин этих неудач, по мнению экспертов, можно назвать следующее:

- недостаточно четкая и полная формулировка требований к ПО;
- отсутствие постоянного контакта с заказчиком;
- недостаточность необходимых ресурсов;
- отвратительное планирование и безграмотное управление проектом;
- частое изменение требований, спецификаций, условий;
- несовершенство, недостаточное знание или новизна используемой технологии;

- отсутствие поддержки со стороны высшего руководства;
- низкая квалификация разработчиков, недостаток необходимого опыта.

Расходы, связанные с проектом

Для разработки ПО, естественно, требуются капиталовложения. На что они расходуются?

На следующие статьи расходов:

- зарплату разработчикам;
- аренду помещения;
- амортизацию средств разработки (компьютеры, сети, программные продукты);
- оплату энергоресурсов;
- оплату управляющего персонала.

Согласно существующей терминологии расходы делятся на прямые и непрямые.

К **прямым расходам** относятся затраты, которые непосредственно переносятся на себестоимость конкретной продукции или услуг. Сюда относятся: материалы, заработная плата основных производителей (программисты, тестировщики, руководители отделов и частей проекта, научные сотрудники и консультанты и т. п.), отчисления на социальные нужды (пенсионный фонд, фонд социального страхования, фонд занятости и фонд обязательного медицинского страхования), электроэнергия, затраты за прошедший период, отнесенные на стоимость продукции

отчетного периода, и прочие прямые затраты. Эти расходы и косвенные расходы характеризуют группу затрат производства по способу включения их в себестоимость продукции: они относятся прямо на счет производства.

Непрямые расходы (косвенные расходы) – это расходы, которые трудно связать с какой-то конкретной деятельностью или проектом, но тем не менее необходимые для нормального функционирования организации и успешного выполнения ее задач: оплата труда администрации, реклама, стоимость износа основных фондов, амортизации капитального оборудования, общие коммунальные расходы (телефон, газ, электричество, лифт, антенна и др.).

(по материалам http://www.prpc.ru)

2. МОДЕЛИ И МЕТОДОЛОГИЯ ПРОЦЕССА РАЗРАБОТКИ

Общий обзор моделей и методологий процесса разработки

Исторически сложились несколько моделей процесса разработки программных систем. Первая, которую называют «классической», пришла из инженерной практики и соответствует процессам, пришедшим из конструирования систем. Все работы, составляющие классическую модель, выполняются последовательно, друг за другом в заранее определенном порядке. К сожалению, такая хорошо понятная модель не работает почти никогда – в процессе разработки постоянно возникают новые требования и уточнения, отказ от которых приводит к конфликту с заказчиком. Разделение процессов кодирования и тестирования приводит к трудновыполнимым переделкам кода. Позже возникли новые виды – итерационные, позволяющие производить корректировки и уточнения многократно и включающие тестирование в процесс разработки с самого ее начала.

Фазы процесса

В любом случае независимо от модели процесс разработки можно разбить на несколько отдельных составных частей, которые называются **фазами процесса проекти-**

рования. Именно последовательность их выполнения и определяет ту или иную модель.

Рисунок 3. Классическая модель. Фазы процесса разработки

Фазы процесса разработки

1. **Определение требований**

Эта фаза необходима для выяснения, чего хочет заказчик и что сделать, чтобы это функционировало. Разрабатываемая система рассматривается как единое целое, определяется функциональность системы, характеристики оборудования, на котором система будет функционировать, физическое размещение системы, первое приближение по определению затрат на разработку. Обычно эта фаза разбивается на две составляющие: системный анализ и анализ требований.

Системный анализ – эта часть фазы определения требований задает роль каждого элемента в компьютерной системе и описывает взаимодействие элементов друг с другом. Он начинается с определения требований к каждому элементу системы и объединения этих требований в программную часть системы. Когда формируется интерфейс ПО с другими составляющими частями системы – аппаратурой, людьми, базами данных и т.п. наиболее ярко проявляется необходимость системного подхода. В то же время решается задача планирования проекта ПО, в ходе которого определяются объем и риск каждой из проектных работ, рассчитываются требуемые трудозатраты, оформляются рабочие задачи и план-график выполнения работ.

Анализ требований – эта часть фазы определения требований относится к программной составляющей – программному обеспечению. При анализе требований производится уточнение и детализация функций ПО, его характеристик и интерфейса. Все произведенные определения сохраняются в документе под названием спецификация анализа. Эта часть фазы определения требований завершает планирование проекта.

2. **Проектирование**

Фаза проектирования определяет то, что станет основой разработки, позволит отобразить:

- архитектуру ПО;
- модульный состав ПО;
- алгоритмическую структуру ПО;
- структуры данных;

- входной и выходной интерфейсы (формы ввода и отчеты).

Исходные данные для проектирования записываются в виде спецификации анализа. В ходе проектирования требования к ПО переводятся в множество проектных представлений (например, множество UML-диаграмм). При проектировании ПО основное внимание уделяется качеству создаваемого программного продукта. Важно правильно определить подсистемы ПО и способы их взаимодействия.

3. **Конструирование («реализация», «кодирование»)**

Обычно именно эту фазу называют программированием, хотя она решает только роль перевода того, что мы уже знаем на какой-либо язык программирования. Все подсистемы, определенные на этапе проектирования, разрабатываются параллельно (если хватает ресурсов: фирма из одного человека не может разрабатывать подсистемы параллельно ☺).

4. **Интеграция**

Если интерфейсы подсистем разработаны в соответствии со спецификацией, подсистемы не оказывают побочного влияния друг на друга и при разработке спецификации учтены все требования – процесс сборки или объединения подсистем не вызывает никаких затруднений. Так никогда не бывает. Во-первых, не все подсистемы разрабатываются одновременно. Во-вторых, ошибки в одной подсистеме часто приводят к неправильному функционированию группы зависящих подсистем.

Поэтому при сборке используют два подхода: метод «большого взрыва» (*сейчас все заработает!*) и метод последовательного подключения. Ни тот, ни другой не позволяют с абсолютной точностью определить подсистему, приводящую к ошибкам. При «большом взрыве» все подсистемы включаются одновременно, количество ошибок зашкаливает и обычно пытаются решить проблему методом последовательного подключения. Однако есть подсистемы, зависящие от еще неподключенных подсистем настолько, что при включении перестают выполнять свои функции, определить первопричину сложно. Поэтому начинается следующая фаза.

5. **Тестирование и отладка («верификация»)**

На фазе тестирования производится поиск дефектов. Для облегчения поиска дефектов разрабатываются тестовые примеры. Успешное или неуспешное выполнение примера называется тестовым случаем. Теоретически невозможно проверить все ветки программного кода, поэтому в действительности количество тестовых примеров и время на тестирование ограничено. Тестирование делится на альфа-тестирование (сам разработчик) и бета-тестирование (заказчик или третьи лица, не являющиеся ни заказчиком, ни разработчиком). Найденные дефекты отправляются на доработку.

6. **Инсталляция**

Эта фаза выявляет недочеты и неувязки, связанные с работой разрабатываемого ПО на оборудовании и в окружении заказчика.

Возможны самые разные неувязки. Назовем некоторые:

- в программе жестко забит путь, которого нет на компьютере заказчика (или хост);
- в программе используется служба, отсутствующая в сети заказчика (например, вы приносите программу на C#, а у заказчика отсутствует или неактуальная версия .Net Framework);
- у заказчика компьютер и операционная система соответствуют требованиям спецификации, но из-за перегруженности компьютера различными фоновыми процессами ваше программное обеспечение не в состоянии поддерживать нужные характеристики интерактивной работы (*случай из практики автора*).

7. **Поддержка**

В процессе эксплуатации заказчиком разработанного ПО могут возникать различные ситуации, которые могут быть разрешены только совместно с разработчиком:

- сменили оборудование (принтер, монитор, мышь, какой-либо адаптер, жесткий диск, процессор и т. п.);
- сменили требования к генерируемым отчетам (новые формы налоговой) или интерфейсу (уберите «пищалку»);
- необходимость обновления базовых справочников, которые могут быть скорректированы только разработчиком.

Все эти вопросы, не решенные своевременно, наносят вред престижу фирмы и количеству возможных

заказчиков (антиреклама). Поэтому исполнитель вносит в план разработки фазу поддержки. Поддержка может выполняться не самим исполнителем а доверенным лицом (например, хорошо известная система 1С).

Кроме описанных фаз возможно наличие и других, но эти семь присутствуют практически во всех моделях разработки. Еще раз перечислим фазы:

1. Определение требований.
2. Проектирование.
3. Конструирование.
4. Интеграция.
5. Тестирование.
6. Инсталляция.
7. Поддержка.

Наличие и взаимное расположение этих фаз а так же добавление некоторых новых фаз или функций, возложенных на фазу, взаимодействие коллектива в процессе выполнения каждой фазы – все это определяет так называемую **модель разработки.**

Попробуем оценить преимущества и недостатки наиболее распространенных моделей.

Водопадная модель

Еще ее называют **каскадной** или **классической**. Автор Уинстон Ройс. Представляет строгое и понятное построение. Нет возврата на предыдущую фазу. Готовый проект заказчик видит только в конце проектирования, когда

у заказчика может поменяться представление о природе приобретаемого ПО.

На рисунке 4 видно, что она отличается от предыдущих другим названием фаз, некоторые фазы собраны в одну, некоторые – разбиты на составляющие.

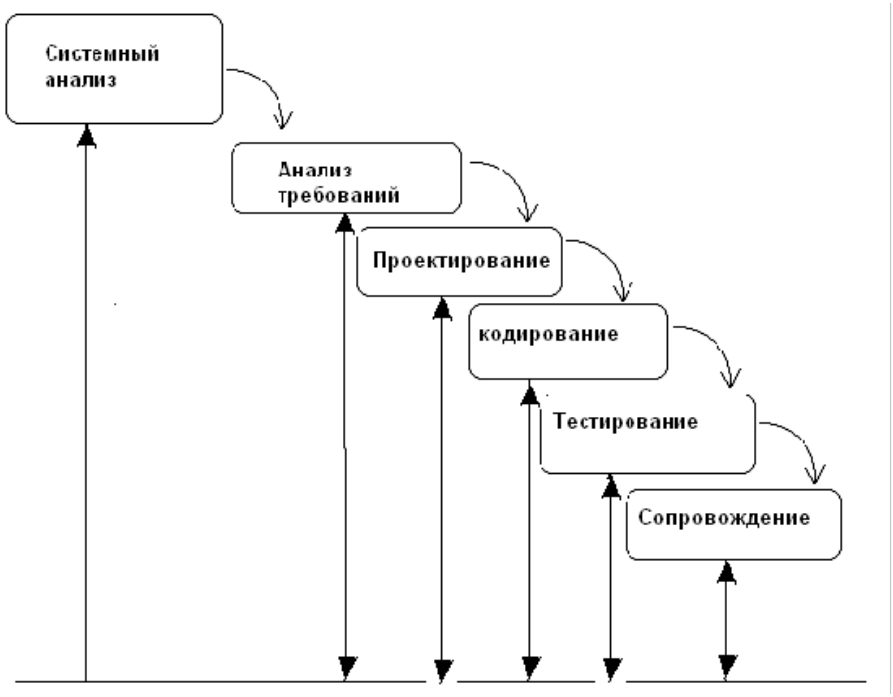

Рисунок 4. Каскадная модель Уинстона Ройса

Это устаревшая, но все еще часто применяющаяся на практике модель. Она предоставляет план и временной график для всех этапов проекта, упорядочивает ход конструирования. Нужно подчеркнуть, что реальные проекты часто требуют изменения стандартной последовательности шагов. Тем не менее, благодаря тому что жизненный цикл разработки основан на точной фор-

мулировке исходных требований к ПО, большинство заказчиков признает именно эту модель, хотя в жизни в начале проекта заказчик может определить свои требования лишь частично. Такая модель имеет наибольшее количество незавершенных проектов.

Макетирование

Макетирование – это не модель, а методология работы с заказчиком, которая с успехом используется в самых различных моделях проектов.

Если заказчик не может сразу сформулировать требования для разрабатываемого ПО или разработчик не уверен в нормальном функционировании ПО в реальной среде заказчика, в реализации диалога с пользователем или в эффективности реализуемого алгоритма принято использовать **макетирование**.

Основная цель макетирования – избавиться от неопределенностей в требованиях заказчика.

Макетирование (*еще его называют прототипированием)* – использование упрощенной модели вместо требуемого программного продукта.

Такая модель принимает одну из трех форм:

- бумажный макет или макет на основе ПК (изображается или рисуется человеко-машинный диалог);
- работающий макет (выполняется некоторая часть требуемых функций);
- существующая программа (характеристики которой затем должны быть изменены).

Макетирование основывается на многократном повторении итераций создания все более точного макета, в которых участвуют заказчик и разработчик.

В принципе, макетирование может применяться в любых моделях разработки ПО как составная часть фазы определения требований.

Достоинства макетирования: макетирование обеспечивает наиболее полное определение требований к ПО.

Недостатки:

- заказчик может решить, что макет – это разрабатываемая система;
- разработчик может решить, что макет – это разрабатываемая система.

Рисунок 5. Обобщенная модель работы с макетом

Спиральная модель

Спиральная модель предложена в 1988 году Барри Боэмом. В основу спиральной модели взяты:

- макетирование;
- сильно измененная водопадная модель, закрученная в эволюционную спираль;
- новая составляющая часть – анализ рисков.

Рисунок 6. Спиральная модель

Спиральная модель:

1 – первичный сбор требований и планирование проекта; 2 – учет рекомендаций заказчика по внесению изменений; 3 – анализ риска на основе первичных требований; 4 – анализ риска на основе реакции заказчика на предыдущий этап; 5 – переход к комплексной системе; 6 – начальный макет системы; 7 – следующий уровень макета системы; 8 – очередная реализация; 9 – оценка реализации заказчиком.

Спиральная модель включает четыре этапа, размещенные в четырех квадрантах по спирали (см. рис. 6):

1. **Планирование** – определение целей проекта или очередного «витка» разработки, возможных вариантов решений и всяческих ограничений.
2. **Анализ риска:** анализ вариантов и выбор – выполнять дальнейшую разработку или риск превышает возможный положительный эффект.
3. **Конструирование** – прерход к следующему уровню разработки.
4. **Оценивание** – тестирование и оценка заказчиком текущих результатов разработки.

Спиральная модель относится к эволюционным моделям, характерная черта которых – многократный «мини» жизненный цикл, с выходом по окончании на новый уровень, новый «виток спирали», с увеличением количества доступных заказчику функций и ростом качества продукта. Заказчик достаточно часто привлекается к процессу разработки и после каждого цикла может видеть и оценить состояние системы.

Достоинства:

- близко к реальности в эволюционном виде отображает процесс разработки программного обеспечения;
- позволяет явно учитывать риски на каждом эволюционном «витке» разработки;
- включает в итерационную модель разработки системный подход (водопадную модель);
- использует макетирование для снижения риска и совершенствования программного изделия.

Недостатки:

- нет достаточной статистики эффективности модели;
- повышаются требования к заказчику;
- возникают проблемы с контролем и управлением временем разработки (заказчику не видно, когда же будет готов конечный продукт, разработчик не знает, обеспечен ли он работой на будущий период).

Компонентно-ориентированная модель

Компонентно-ориентированная модель – это развитие спиральной модели. Она тоже основывается на эволюционной стратегии конструирования. Здесь уточняется содержание квадранта конструирования: отражается тот факт, что в современных условиях новая разработка должна основываться на повторном использовании существующих программных компонентов (см. рис. 7 на следующей странице).

Итеративная модель

В принципе, спиральная модель и все эволюционные модели является разновидностью итеративной модели. Итеративная модель указывает на повторение группы действий и вносит в процесс разработки анализ риска после каждой итерации. Кроме спиральной модели существует множество методик, суть которых сводится к минимизации риска за счет выполнения в виде множества коротких циклов.

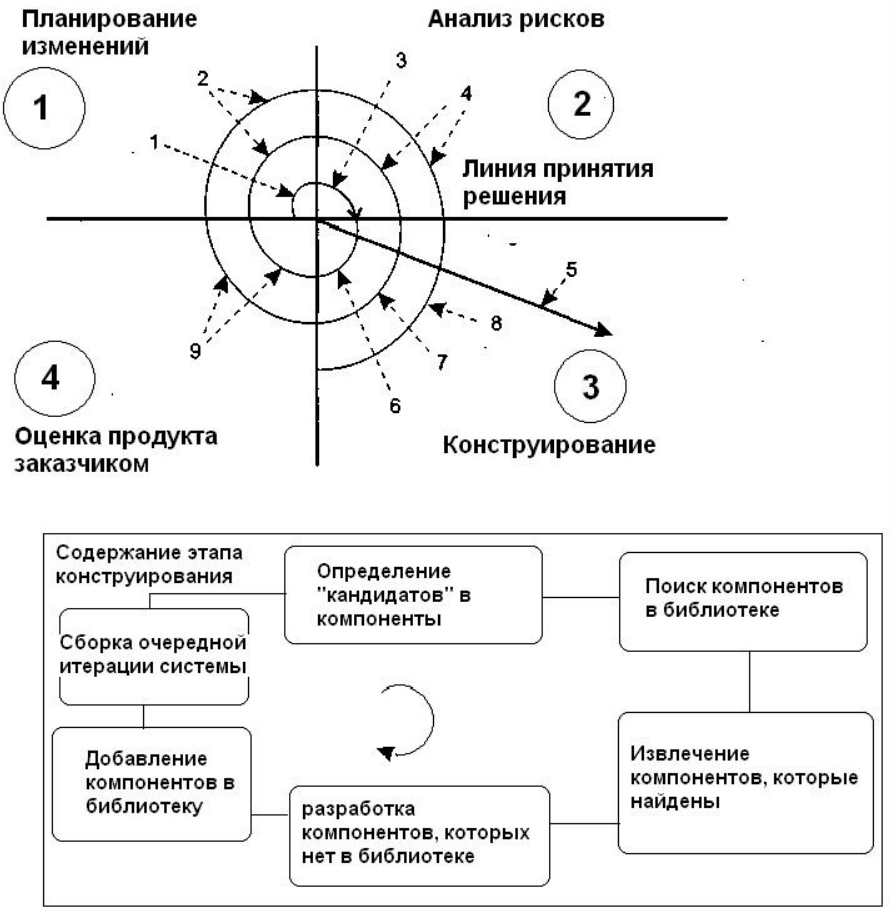

Рисунок 7. Компонентно-ориентированная модель

Назовем некоторые из них.

Agile

Agile – семейство методов разработки в разработке программного обеспечения, которое началось со спиральной модели (1985 г.) – многократно повторяющийся короткий, но полный цикл разработки.

Следующим этапом развития Agile *(эджайл)* стала методология **Scrum**, опирающаяся на 30-дневные циклы, самоорганизацию команд разработчиков и ежедневные митинги (1990–1999 гг.). Еще один вид самоадаптирующегося итерационного процесса разработки был предложен авторами **RAD** (Rapid Application Development) в 1994-м. К 96-му году относят описание методологии Кента Бека **XP** (eXtreme Programming), вариант итерационного процесса разработки с гипертрофированным тестированием и передачей кода между членами команды (совместное владение кодом). И, наконец, последняя из вошедших в Agile итеративных методик – Feature Driven Development (**FDD**), предложенная в 1997 году и сразу спасшая безнадежный проект.

Авторы перечисленных методологий опубликовали их обобщенные черты в ***Agile Manifesto*** – основном документе для процессов гибкой, допускающей изменения требований и даже целей разработки. ***Agile Manifesto*** определяет ценности и принципы, которые применяют наиболее успешные команды.

Подход включает ценности, значимость которых в неуспешных командах сведена к нулю:

- процессы и инструменты не так важны, как личности и их взаимодействия;
- работающее программное обеспечение более важно, чем полная документация;
- реакция на изменения более важна, чем неукоснительное исполнение плана. Сотрудничество с заказчиком более важно, чем обязательства по контракту.

Вот принципы, которые включены в ***Agile Manifesto***:

- ежедневное общение заказчика с разработчиком на протяжении реализации проекта;
- приветствуется изменения требований даже в конце разработки, что повышает конкурентоспособность полученного продукта;
- частая поставка рабочего ПО заказчику (возможно, каждый месяц или неделю);
- лучшее удовлетворение клиента за счет как можно более ранней и бесперебойной поставки ценного ПО;
- все действующие лица проекта – спонсоры, разработчики и пользователи – должны иметь возможность поддерживать постоянный темп на длительный срок;
- простота разработки – выполнять разработку только действительно необходимых частей;
- проектом должны заниматься мотивированные личности, которых необходимо обеспечить нужными условиями работы, поддержкой и доверием;
- наиболее рекомендуемый способ общения – личный разговор (лицом к лицу);
- работающее ПО – лучший измеритель прогресса разработки;
- постоянное внимание уделяется улучшению технического мастерства и удобству дизайна;
- частая или постоянная адаптация (улучшение эффективности работы) к изменяющимся условиям;
- у самоорганизованной команды получаются лучшие архитектура, требования и дизайн.

Далее рассмотрим некоторые Agile-методологии более подробно.

Scrum

Scrum *(скрам)* – методология управления проектами для гибкой разработки программного обеспечения. Авторы – Джеф Сазерленд и Кен Швабер. Методология основана на анализе практик японских промышленных компаний и ежедневных митингах фирмы «Борланд».

Процесс разработки делится на 30-дневные циклы ***(спринты***). В начале процесса разработки определяются все требования и цели и выбираются приоритетные для первого цикла. В конце каждого цикла команда на собрании ***(митинге, ревью)*** предоставляет руководству и заказчику готовый к использованию продукт, в котором пока учтены не все требования заказчика, а только та функциональность, которая была запланирована на этот спринт. Заказчик вместе с менеджером проекта рассматривает текущее состояние продукта, оценивает его. По итогам митинга принимающая сторона может сделать выводы о том, как развивается система, что можно улучшить, а также выбирает, что нужно сделать в следующем спринте.

Руководство не вмешивается в работу команды на протяжении всего цикла (*полное доверие*) и не добавляет команде дополнительных задач после утверждения задач на спринт. Для «утрясания» взаимодействия команды и неукоснительного соблюдения правил скрама выбирается скрам-мастер. Команда ежедневно производит митинги-отчеты каждого члена команды и решает о пе-

перераспределении усилий для своевременного решения поставленного круга задач, что является глубокой обратной связью и позволяет качественно контролировать процесс разработки.

С точки зрения методологии, Scrum включает набор методов, предварительно определенных ролей и артефактов (в данном случае – контрольных точек, определяющих направление развития, к примеру, артефактом может быть доказанная неприменимость одного из подходов). Полученные артефакты обсуждаются на митингах.

По окончании спринта производится ***демонстрация*** и ***ретроспектива*** (совместный митинг команды, руководства и заказчика)

Более подробно о методологии Scrum вы узнаете в дальнейших уроках.

Терминология Scrum

Роли:

- Владелец Продукта **(Product Owner)**
- Руководитель **(ScrumMaster)**
- Команда **(Scrum Team)**
- Пользователи **(Users)**
- Клиенты, Продавцы **(Stakeholders)**
- Эксперты-консультанты **(Consulting Experts)**

Методы:

- Планирование спринта **(Planning Meeting)**
- Остановка спринта **(Sprint Abnormal Termination)**
- Митинг **(Daily Scrum)**

- Демонстрация **(Demo Meeting)**
- Ретроспектива **(Retrospective Meeting)**

Артефакты:

- Product backlog
- Sprint Backlog

XP

Узнаете? Да это же устаревшая версия Windows! ☺ Конечно же нет! Аббревиатура возникла от **Extreme Programming.** Вся команда программистов садится в гамак или становится на лыжи, можно – скейты, и… Нет.

Идея экстремального программирования – доведение всех составляющих гибкого (*Agile*) программирования до абсурда. Если цикл разработки должен быть коротким, то выбираем как можно более короткий цикл. Если встречи с заказчиком должны производиться часто – давайте включим представителя заказчика в команду. Если тестирование нужно производить как можно чаще – давайте разрабатывать функцию продукта только после разработки теста для нее. Если не следует перегружать продукт лишней функциональностью – давайте определять через заказчика только те функции, которые ему крайне необходимы. Если заказчик сомневается в необходимости дальнейшей разработки – давайте сразу же прекращать ее. Если есть трудность в овладении программистом чужого кода – сделаем так, чтобы код разрабатывался не одним, а сразу двумя программистами (парное программирование, смена программистов в парах на протяжении проекта). Если человек не лошадь и мы хотим

максимальной производительности и сосредоточенности на протяжении всего процесса разработки – 40-часовая рабочая неделя.

RUP

Rational Unified Process (RUP) – это метод разработки программного обеспечения, созданный компанией Rational Software, и эта же компания обеспечила свой метод полным набором инструменнтов.

Прежде всего, RUP – итеративная модель разработки. В конце каждой итерации (от 2 до 6 недель) разработчики должны достичь запланированных на данную итерацию целей, создать или доделать проектные артефакты и получить промежуточную функциональную версию программного обеспечения.

Такой способ разработки позволяет быстро реагировать на изменившиеся требования, обнаружить и устранить риски на ранних стадиях проекта, а также высокоэффективно контролировать качество создаваемого продукта.

Итерация разработки разделяется на 4 фазы:

1. Начало **(Inception).**
2. Проектирование **(Elaboration).**
3. Построение **(Construction).**
4. Внедрение **(Transition).**

Каждая итерация завершается контрольной точкой, позволяющей оценить успешность этой итерации.

В процессе разработки контролируется множество процессов (для каждого имеется соответствующий инструмент):

- моделирование бизнес-процессов;
- управление требованиями;
- анализ и проектирование;
- реализация;
- тестирование;
- развертывание;
- конфигурационное управление и управление изменениями;
- управление проектом;
- управление средой.

MSF

Microsoft Solutions Framework (MSF) – это методология разработки программного обеспечения от Microsoft, фактически, вариант спиральной модели.

Спираль разделена на этапы контрольными точками:

1 этап – создание общей картины;

1 контрольная точка – утверждение документа общей картины;

2 этап – планирование;

2 контрольная точка – утверждение проектных планов;

3 этап – разработка;

3 контрольная точка – окончательное утверждение области действия проекта;

4 этап – стабилизация (тестирование и подготовка к развертыванию);

4 контрольная точка – подтверждение готовности проекта к выпуску;

5 этап – развертывание;

5 контрольная точка – решение развернуто.

Используется богатый практический опыт «Майкрософт». Включает описание принципов управления людьми и рабочими процессами.

MSF использует две разновидности моделей:

- модель проектной группы;
- модель процессов.

MSF включает 3 дисциплины:

- дисциплина управление проектами;
- дисциплина управление рисками;
- дисциплина управление подготовкой.

Microsoft не использует в «чистом варианте» саму методику MSF в своих IT-проектах.

Анализ существующих моделей и методов

В последние годы успешность проекта занимает все большую значимость как для создателей, так и для заказчиков проекта. Практика показывает, что нет возможности спланировать и реализовать долгосрочный проект без внесения изменения на протяжении всего процесса. Поэтому возникают новые методологии, некоторые из них больше подходят для небольших коллективов разработчиков и мелких заказчиков – риски не так велики, нет

ограничений на время разработки, нечетко определены цели разработки (спиральная модель и XP, возможно – Scrum). На практике большинство итерационных методов конкурируют друг с другом, но некоторые (RUP, MSF) позволяют все же выполнять разработку больших проектов с определенными целями, временем разработки и бюджетом, что применимо для больших коллективов разработчиков, так как заранее определен объем работ и, соответственно, обеспечена занятость.

Соответственно первые, итерационные методы разработки получили название «легковесных» *(lightweight),* а вторые, требующие полного определения всех характеристик продукта на начальном этапе – «тяжеловесные» *(heavyweight).*

3. УПРАВЛЕНИЕ КАЧЕСТВОМ

Для любого предприятия главным источником существования является успешная реализация качественного продукта.

Говоря о качестве продукта, всегда подразумевается потребитель, потому что именно потребитель определяет приемлемые свойства товара.

В условиях рыночной экономики качество продукта это задача номер один, так как конкурентоспособность фирмы определяется:

1. Уровнем цены продукции.
2. Уровнем качества продукции.

Причем качество продукции постепенно выходит на первое место.

Параметров, определяющих качество товара, большое множество, поэтому в менеджменте возникла необходимость развития такого течения, как управление качеством.

> ***Управление качеством*** *– деятельность по управлению всеми этапами жизненного цикла продукции, а также взаимодействием с внешней средой.*

В настоящий момент во всем мире разработано множество (несколько сотен) систем контроля качества продукции. Общая задача этих систем – соответствие продукции требованиям потребителей.

Наиболее известным и широко используемым стандартом для организации процессов контроля качества является серия стандартов **ISO 9000**.

Для разработки программного обеспечения используется руководство ISO 9000–3, в котором содержится описание процесса разработки для достижения нужного уровня качества.

В настоящее время рекомендуется использование стандарта версии 2014 года. Главное внимание теперь уделяется управлению процессом. Пока стандарт не содержит части для разработчиков ПО.

Внедрение стандартов ISO 9000 создает базу для независимой сертификации продукции, которая ориентирована на потверждение уровня качества продукции, определяющего ее конкурентные возможности.

Недостаток стандарта ISO 9000 – нет возможности адекватно оценить уровень качества процесса разработки ПО для предложенной модели оценки качества.

Альтернативная модель качества (США): **CMM – SEI (Software Engineering Institute).** Эта модель качества разработана в институте инженерии программного обеспечения для использования государственными, в частности, военными организациями при размещении заказов на разработку программного обеспечения.

Сейчас модель CMM – SEI применяется для анализа и сертификации процессов разработки программного обеспечения фирмами, производящими наиболее сложные разработки в программировании.

История развития систем качества

В истории развития систем качества можно выделить следующие пять этапов:

- **1905 г.: Система Тейлора** – система требований к качеству изделий в виде допусков (предельное минимальное и максимальное значения). Содержит шаблоны, настроенные на верхнюю и нижнюю границы допусков. Это система управления качеством каждого отдельного взятого изделия;
- **1924 г.:** Фирмой Bell Telephone Laboratories заложены **основы статистического управления качеством**. Позже статистические методы контроля качества получили широкое распространение. Для Японии статистические методы контроля стали основой экономической революции. При статистических методах контроля качества вместо проверки и выявления дефектов главное внимание уделяется предупреждению дефектов путем устранения их причин;
- **1950-е гг.:** Появилась идея **тотального (всеобщего) контроля качества – TQC (Total Quality Control)**. Автор – американский ученый А. Фейгенбаум (*1957 г., статья «Комплексное управление качеством»*). Главные задачи TQC:
  1. Предсказуемое исправление возможных несоответствий продукции еще на стадии конструирования.
  2. Проверка качества поставляемой продукции, комплектующих и материалов.
  3. Управление производством.
  4. Применение службы сервисного обслуживания.

5. Наблюдение за соответствием предъявленным требованиям.

На этом этапе появились документированные системы качества, устанавливающие ответственность, полномочия и взаимодействие в области качества руководства предприятия и специалистов служб качества. В Европе появились аудиторы (независимая третья сторона, выполняющая регистрацию и сертификацию систем) и большое внимание стали уделять документированию систем обеспечения качества. Япония встретила с восторгом идеи TQC и предприняла шаги для их дальнейшего развития.

- **80-е гг.:** происходит **переход от тотального контроля качества (TQC) к тотальному управлению качеством (TQM).**

Появилось множество новых международных стандартов на системы управления качеством – стандарты ISO 9000 (1987 г.). TQM не просто управляет качеством, TQM предусматривает управление и целями, и требованиями. В TQM обеспечение качества понимается как система мер, призванная вызывать у заказчика программного продукта уверенность в его качестве. Основной принцип TQM – улучшению нет предела. Система управления качеством не решает всех задач, необходимых для обеспечения конкурентоспособности, но она становится все популярнее и сегодня она занимает прочное место в рыночном механизме.

- **90-е гг.: усиливается влияние общества на предприятия, а предприятия все больше учитывают интересы общества**. Это приводит к появлению

стандартов серии ISO 14000, котрые устанавливают требования к системам менеджмента с точки зрения защиты окружающей среды и безопасности продукции.

Появляющиеся стандарты дополняют друг друга, расширяют и приводят к многогранному рассмотрению вопроса управления качеством продукции. Теперь управление качеством – это не просто контроль качественных параметров и причин их отклонений, это управленческая деятельность, которая охватывает жизненный цикл продукции, системно обеспечивает стратегические и оперативные процессы повышения качества продукции и функционирование самой системы управления качеством.

Что же такое качество? Факторы качества

Качество – мера полезности, целесообразности и эффективности труда, ощущаемая каждым человеком. Повышение качества, в свою очередь, приводит к снижению потерь на всех этапах жизненного цикла продукции (маркетинг > разработка > производство > потребление > утилизация). В результате этого снижается себестоимость и цена продукта, а это приводит к повышению жизненного уровня людей. Сейчас существует несколько определений качества, которые в целом совместимы друг с другом.

Определение «качество ПО» в международных стандартах звучит так:

> ***Качество программного обеспечения*** – *совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.*
>
> [ISO 8402:1994 Quality management and quality assurance]

Качество программного обеспечения *– степень, в которой ПО обладает требуемой комбинацией свойств.*

[1061–1998 IEEE Standard for Software Quality Metrics Methodology]

Согласно *wikipedia.org*:

*«**Фактор качества ПО** – это нефункциональное требование к программе, которое обычно не описывается в договоре с заказчиком, но, тем не менее, является желательным требованием, повышающим качество программы».*

(wikipedia.org)

В таблице ниже приведены различные факторы качества:

Понятность	Назначение ПО должно быть понятным – программный интерфейс и документация
Полнота	Программа должна быть реализована полностью
Краткость	Отсутствие дублирующейся информации. Отсутствие дублирующего кода (замена на вызов процедур). Отсутствие дублирования документации
Портируемость	Легкость переноса ПО в новое окружение
Согласованность	Во всей программе и документации должны использоваться одинаковые форматы и обозначения
Сопровождаемость	Требование к документации и наличию резерва роста при изменении ресурсов (память, процессор). Насколько просто изменить программу при изменившихся требованиях заказчика
Тестируемость	Программа должна выполнять тестовые примеры и средства для измерения производительности
Удобство использования	Простота и удобство интерфейса пользователя

Надежность	Отсутствие отказов и сбоев в работе программ, а также простота восстановления рабочего состояния
Структурированность	Четко выделенные части программы и связи между ними
Эффективность	Рациональное использование ресурсов
Безопасность	Хранение и ограничение доступа к конфиденциальной информации, фиксация транзакций

Управление и обеспечение качества – понятие, которое охватывает все этапы разработки, выпуска и эксплуатации ПО.

4. ДОКУМЕНТИРОВАНИЕ

Документация на программное обеспечение – это документы, в которых описаны правила установки и эксплуатации ПО.

Документирование ПО является настолько важным этапом его разработки, что зачастую успех распространения и эксплуатации программного продукта в большой степени зависит именно от качества его документации. Чем сложнее и запутаннее проект, тем выше роль сопровождающих документов. Необходимо отметить, что типы документов бывают различны в зависимости от характера информации, которую они содержат.

Можно отметить пять целей разработки документов:

- документы устанавливают связи между всеми вовлеченными в процесс разработки;
- документы описывают обязанности группы разработки;
- документы позволяют руководителям оценивать ход разработки;
- документы образуют основу документации сопровождения программного обеспечения;
- документы описывают историю разработки программного обеспечения.

Качество программного обеспечения, наряду с другими факторами, определяется полнотой и качеством пакета документов, сопровождающих ПО. В пакет документов, сопровождающих ПО включаются документы, содержа-

щие сведения, необходимые для разработки, изготовления, сопровождения и эксплуатации программы.

Существует четыре основных типа документации на ПО:

- **архитектурная** или **проектная** – общее описание программного обеспечения, рабочей среды и основных идей создания ПО;
- **техническая** – подробная документация на программный код, используемые алгоритмы, межмодульные интерфейсы;
- **пользовательская** – руководства для конечных пользователей, администраторов системы, монтажников и др.;
- **маркетинговая** – принципы реализации, бонусные и партнерские дополнения и т. п.

Любая компания предъявляет уникальные требования к составу и содержанию документов для своей продукции. Большинство этих требований основано на положениях и рекомендациях национальных стандартов и/или стандартов, распространенных в мировой практики. Принципы управления документированием программного обеспечения одинаковы для любого объема проекта. Хотя для небольших проектов значительную часть положений можно не применять, но принципы остаются теми же.

Большинство проектов разрабатывается на основе 2–3-х документов:

- техническое задание;
- руководство пользователя;
- руководство администратора.

По требованию заказчика разработчик должен предоставить полный перечень документов, соответствующий государственным или международным стандартам или требованиям контракта.

Документация может быть разработана как для всего программного комплекса, так и для отдельных его составляющих частей.

Перечислим необходимые и достаточные документы, чтобы программная система была создана с необходимым уровнем качества. При этом нужно отметить:

- **необходимость документа** – документ, без которого невозможно создать качественный продукт;
- **достаточность документа** – если работы выполняются в соответствии с документом, гарантированно получится продукт.

Процесс разработки программного продукта похож на многие другие инженерные задачи. Институт инженеров по электротехнике и радиоэлектронике (IEEE, www.ieee.org) разработал стандарты документации процесса разработки программного обеспечения.

Состав документации проекта согласно этим стандартам по группам документов в перечисленном порядке:

1. **План экспертизы программного обеспечения (SVVP – Software Verification and Validation Plan):** документ описывает, каким образом и в какой последовательности должны проверяться стадии проекта и сам продукт на соответствие требованиям.

Проще говоря, этот документ оговаривает то, как заказчик и разработчик убедятся в том, что они сделали то,

что хотели сделать и что они действительно это сделали. Он не описывает, что и как они будут делать.

2. **План контроля качества программного обеспечения (SQAP – Software Quality Assurance Plan):** каким образом проект должен достигнуть соответствия установленному уровню качества.

То есть разработчик в отдельном документе описывает, что он будет стараться, чтобы в программном продукте не было ошибок. А если, не смотря на все «организационные меры», они все-таки возникли, то заказчик будет знать, что разработчик очень старался, чтобы их не было.

3. **План управления конфигурациями программного обеспечения (SCMP – Software Configuration Management Plan):** содержит описание, где и каким образом разработчик будет хранить версии документации, программного кода и каким образом будет определять, как код связан с документацией.

Например, если программисту необходимо внести изменения в программный модуль, ему бывает трудно найти описание того, что именно должен делать этот модуль, что он делал раньше и так далее среди огромного вороха различных версий документов. Для помощи в поисках нужен еще один документ, который описывает, что именно он должен делать, чтобы найти нужную информацию.

4. **План управления программный проектом (SPMP – Software Project Management Plan):** описывает, каким образом разработчик будет управлять проектом создания программного продукта, чтобы довести разработку до качественного результата. Он

включает управления рисками, календарный план, кадровые вопросы и так далее.

5. **Спецификация требований к программному обеспечению (SRS – Software Requirements Specification)**: важнейший документ, который включает две части: требования заказчика и наброски реализации – то, что может быть выполнено.

Поэтому заказчику предоставляют обе части для сверки: правильно ли описаны все его требования в обеих частях документа. После того как заказчик подписал этот документ (приговор), уже он будет виноват в том, если что-либо упустил или передумал; в том, что разработчик ошибся при записи требований или продукт не соответствует желаниям заказчика.

6. **Проектная документация программного обеспечения (SDD – Software Design Document)**: это архитектура ПО и детали проектирования приложения. Не предоставляется заказчику. Зачастую является секретом предприятия.
7. **Документация по тестированию программного обеспечения (STD – Software Test Documentation):** содержит описание того, как, кем и когда должно проводиться тестирование.

Этот подход к разработке документации считался единственно верным в 80–90-х годах XX столетия.

Уже в конце 90-х разработчики пришли к пониманию того, что в программное обеспечение постоянно вносятся изменения, из-за смены требований, и нет необходимости пытаться создать самую долговечную программу.

В результате возник подход, названный экстремальным программированием (XP – eXtreme Programming).

Экстремальное программирование позволяет заказчику:

1. Не определять абсолютно всех требований заранее.
2. Вносить изменения в процессе разработки.
3. Управлять необходимостью дальнейшей разработки ПО.
4. Не оплачивать ненужную документацию.

Экстремальное программирование предъявляет достаточно жесткие требования и к программистам, и создаваемому им программному коду, потому что именно код является основной документацией проекта и его достаточно для успешного сопровождения продукта. Экстремальное программирование требует нескольких составляющих для документирования кода:

- выбор понятных имен функций, переменных и классов;
- модульные интуитивно понятные тестовые случаи;
- развернутые комментарии при описаниях классов и их составляющих.

Стандартизация документации

Основная функция технической документации – хранение и получение всевозможных сведений технического характера. Поэтому техническая документация должна быть как можно более понятной, информативной, удобной и т. п. Наверное, все читают именно такую документацию? Написать понятную и четкую документацию

очень сложно, но существуют стандарты и методики, упрощающие процесс.

У технической документации есть еще одна важная функция – нормативная: она фиксирует **взаимные обязательства участников разработки**. После утверждения технической документации, изменение содержимого в одностороннем порядке юридически невозможно ни заказчиком, ни разработчиком.

Пример 1.

Если в утвержденном техническом задании (ТЗ) на автоматизированную систему сказано, что посетитель сайта должен видеть лозунг «Слава труду», заказчик не имеет права вдруг потребовать еще и демонстрации рекламного ролика на эту тему. Вполне возможно, что показывать ролик – неплохая идея, но заказчику следовало высказать ее раньше, когда разработчик согласовывал с ним техническое задание. С другой стороны, если вместо положенного текста система будет выдавать рекламу ночного клуба, у заказчика появятся все основания потребовать от разработчика внесения в систему необходимых изменений, и не платить ему денег, пока система не будет приведена в соответствие техническому заданию.

Пример 2.

Если в утвержденном техническом проекте зафиксировано, что для реализации автоматизированной системы используется скрипт на PHP, разработчик не имеет права огорошить заказчика предложением вместо бесплатного интерпретатора этого языка приобрести Lotos Domino.

Но, с другой стороны, и заказчик не может потребовать от разработчика вместо нормального сервера использовать компьютер «БК-0010», завалявшийся в кладовке с советских времен.

Документация по эксплуатации является частью продукта, так как без нее невозможно или сильно затруднено применение ПО. Стандартизация документации по эксплуатации позволяет не описывать в договоре или в техническом задании подробные требования к ней. Вместо этого используются ссылки на соответствующие стандарты или их части.

Поэтому разработка комплекта документации должна выполняться специалистом высокой квалификации, который разбирается и в содержимом проекта, и в психологии будущего пользователя. Желательно владение литературным стилем и т. п.

Урок №1

ВВЕДЕНИЕ В УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ