Экономика программной инженерии

Барышникова Марина Юрьевна
МГТУ им. Н.Э. Баумана

baryshnikovam@mail.ru

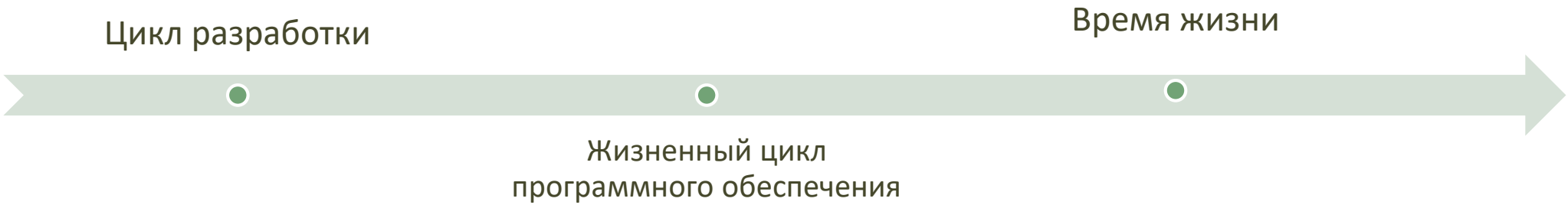
Лекция 3

Жизненный цикл программного обеспечения и его модели. Модели качества программного обеспечения. Модель Фазы – Функции: Модель Гантера

Успех проекта по разработке ПО зависит не только от получения удачного программного продукта, но также от получения удачных процессов его создания

Основные виды деятельности, выполняемые при создании ПО

- проектирование — выделение отдельных модулей и определение связей между ними с целью минимизации зависимостей между частями проекта
- кодирование — разработка кода отдельных модулей
- тестирование — проверка работоспособности программной системы



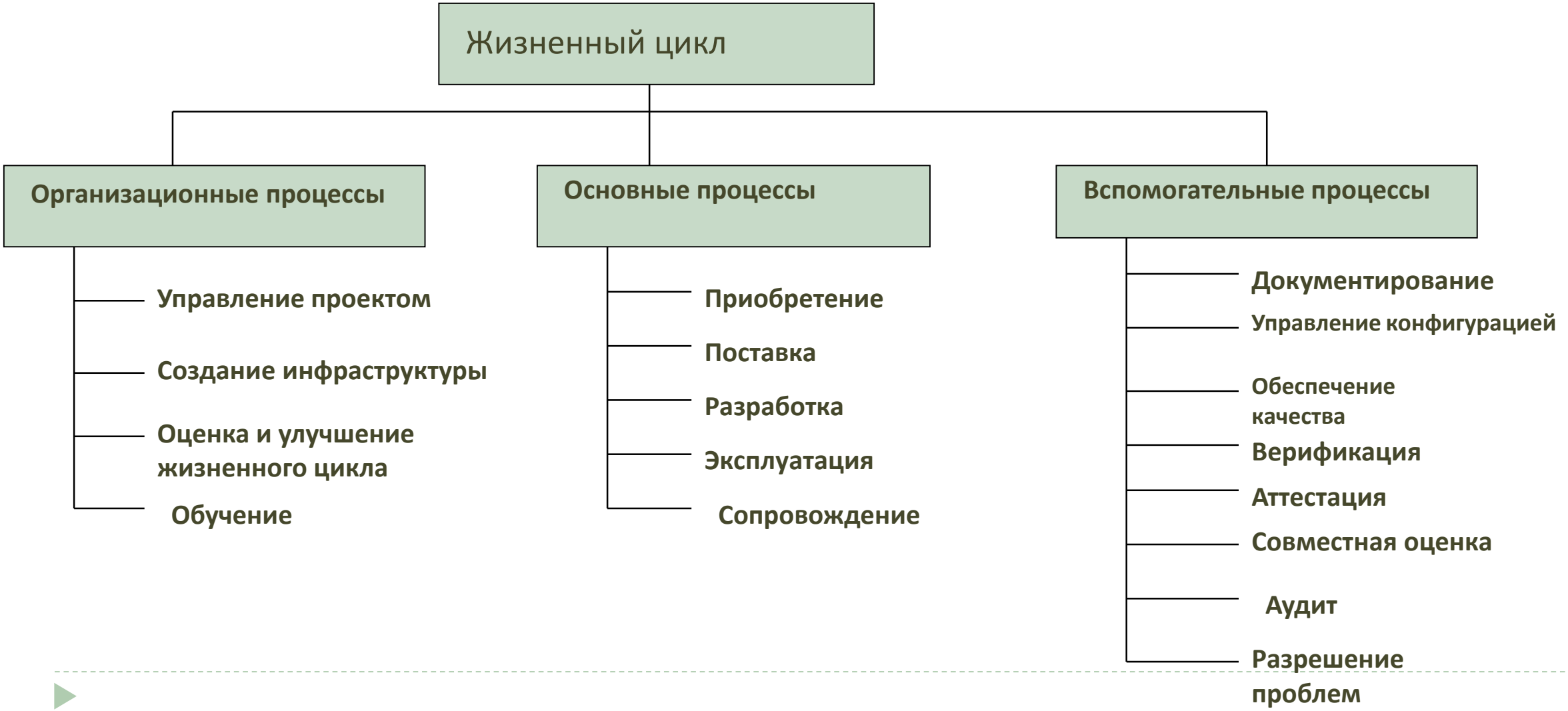
Жизненный цикл программного обеспечения – это период времени, который начинается с момента принятия решения о необходимости создания программного обеспечения и заканчивается в момент его полного изъятия из эксплуатации

(IEEE Std. 610.12 – 1990 Standard Glossary of Software Engineering Terminology)

Основные понятия, участвующие в определении жизненного цикла

- **Артефакты** — создаваемые человеком информационные сущности – документы, в достаточно общем смысле участвующие в качестве входных данных и получающиеся в качестве результатов различных деятельностей.
- **Роль -** абстрактная группа заинтересованных лиц, участвующих в деятельности по созданию и эксплуатации системы, решающих одни и те же задачи или имеющих одни и те же интересы по отношению к ней
- **Программный продукт** – набор компьютерных программ, процедур и, возможно связанных с ними документации и данных
- **Процесс** – совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные

Жизненный цикл ПО согласно стандарту ISO/IEC 12207: 1995 «International Technology – Software Life Cycle Processes» (ГОСТ ИСО МЭК 12207-99 Информационные технологии. Жизненный цикл программного обеспечения)



Содержание основных процессов ЖЦ ПО (ISO/IEC 12207) на примере информационной системы (ИС)

Процесс (исполнитель процесса)	Действия	Вход	Результат
Приобретение (Заказчик)	Инициирование	Решение о начале работ по внедрению ИС	Технико-экономическое обоснование внедрения ИС
	Подготовка заявочных предложений	Результаты обследования деятельности заказчика	Техническое задание на ИС
	Подготовка договора	Результаты анализа рынка / тендера	Договор на поставку / разработку
	Контроль деятельности поставщика	План поставки / разработки	Акты приемки этапов работы
	Приемка ИС	Комплексные испытания ИС	Акт приемно-сдаточных испытаний

Содержание основных процессов ЖЦ ПО (ISO/IEC 12207)

Процесс (исполнитель процесса)	Действия	Вход	Результат
Поставка (Разработчик)	Инициирование	Техническое задание на ИС	Решение об участии в разработке
	Ответ на заявочные предложения	Решение руководства об участии в разработке	Коммерческие предложения / конкурсная заявка
	Подготовка договора	Результаты тендера	Договор на поставку / разработку
	Планирование исполнения	План управления проектом	Реализация / корректировка
	Поставка ИС	Разработанная ИС и документация к ней	Акт приемно-сдаточных испытаний

Содержание основных процессов ЖЦ ПО ИС (ISO/IEC 12207)

Процесс (исполнитель процесса)	Действия	Вход	Результат
Разработка (Разработчик)	Подготовка	Техническое задание на ИС	Используемая модель ЖЦ, стандарты разработки
	Анализ требований к ИС	Техническое задание на ИС, модель ЖЦ	План работ
	Проектирование архитектуры ИС	Техническое задание на ИС	Состав подсистем, компоненты оборудования
	Разработка требований к ПО	Подсистемы ИС	Спецификации требований к компонентам ПО
	Проектирование архитектуры ПО	Спецификации требований к компонентам ПО	Состав компонентов ПО, интерфейсы с БД, план интеграции ПО

Содержание основных процессов ЖЦ ПО ИС (ISO/IEC 12207)

Процесс (исполнитель процесса)	Действия	Вход	Результат
Разработка (Разработчик)	Детальное проектирование ПО	Архитектура ПО	Проект БД, спецификации интерфейсов между компонентами ПО, требования к тестам
	Кодирование и тестирование ПО	Материалы детального проектирования	Тексты модулей ПО, акты автономного тестирования
	Интеграция ПО и квалификационно е тестирование ПО	План интеграции ПО, тесты	Оценка соответствия ПО требованиям ТЗ
	Интеграция ИС и квалификационное тестирование ИС	Архитектура ИС, ПО, документация на ИС, тесты	Оценка соответствия ПО, БД, технического комплекса и комплекта документации требованиям ТЗ

Модель жизненного цикла ПО

Модель жизненного цикла программного обеспечения представляет собой **совокупность упорядоченных во времени, взаимосвязанных и объединенных в стадии работ, выполнение которых необходимо и достаточно для создания ПО, соответствующего заданным требованиям**

Модель жизненного цикла зависит от специфики, масштаба и сложности проекта, а также от условий, в которых система создается и функционирует. Каждая модель ЖЦ увязывается с конкретными методиками разработки программных систем и соответствующими стандартами в области программной инженерии

Организационные аспекты, подлежащие рассмотрению при формировании ЖЦ ПО:

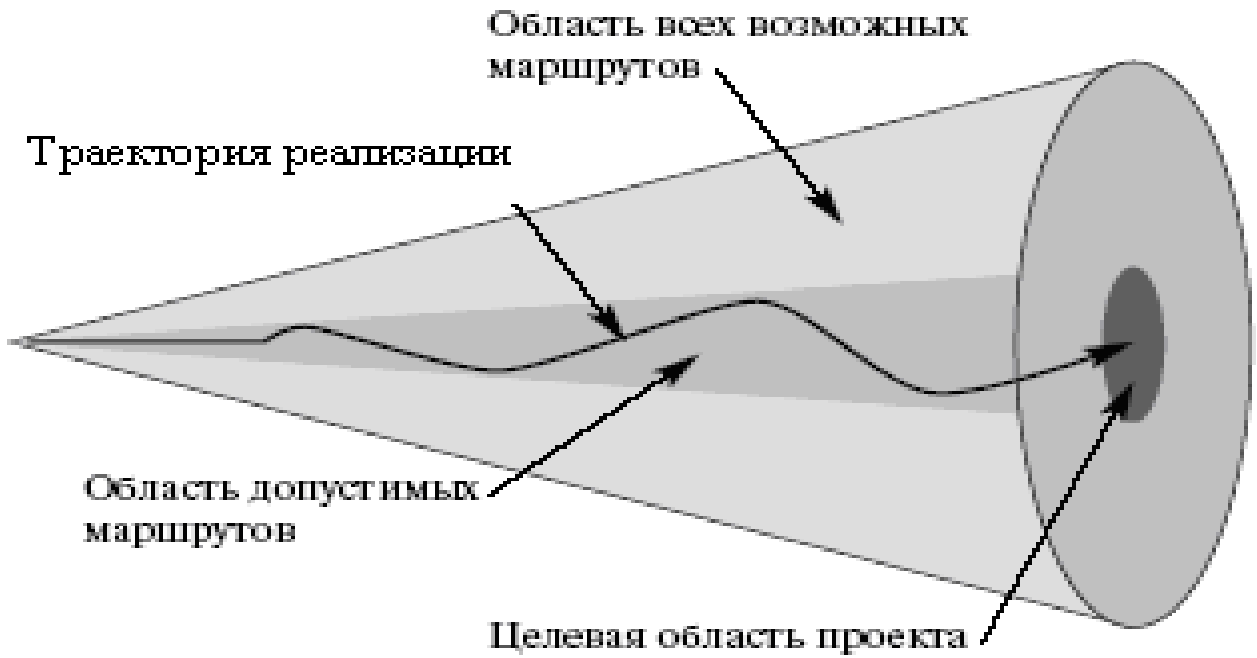
- планирование последовательности работ и сроков их исполнения
- подбор и подготовка ресурсов (людских, программных и технических) для выполнения работ
- оценка возможностей реализации проекта в заданные сроки, в пределах указанной стоимости и др.

Назначение моделей жизненного цикла ПО

- Модель жизненного цикла дает рекомендации по организации процесса разработки ПО в целом, конкретизируя его до видов деятельности, артефактов, ролей и их взаимосвязей
- Модель жизненного цикла служит основой для планирования программного проекта, позволяет экономнее расходовать ресурсы и добиваться более высокого качества управления
- Знание технологических функций, которые на разных этапах должны выполнять разработчики, занимающие те или иные роли, способствует правильному распределению обязанностей сотрудников

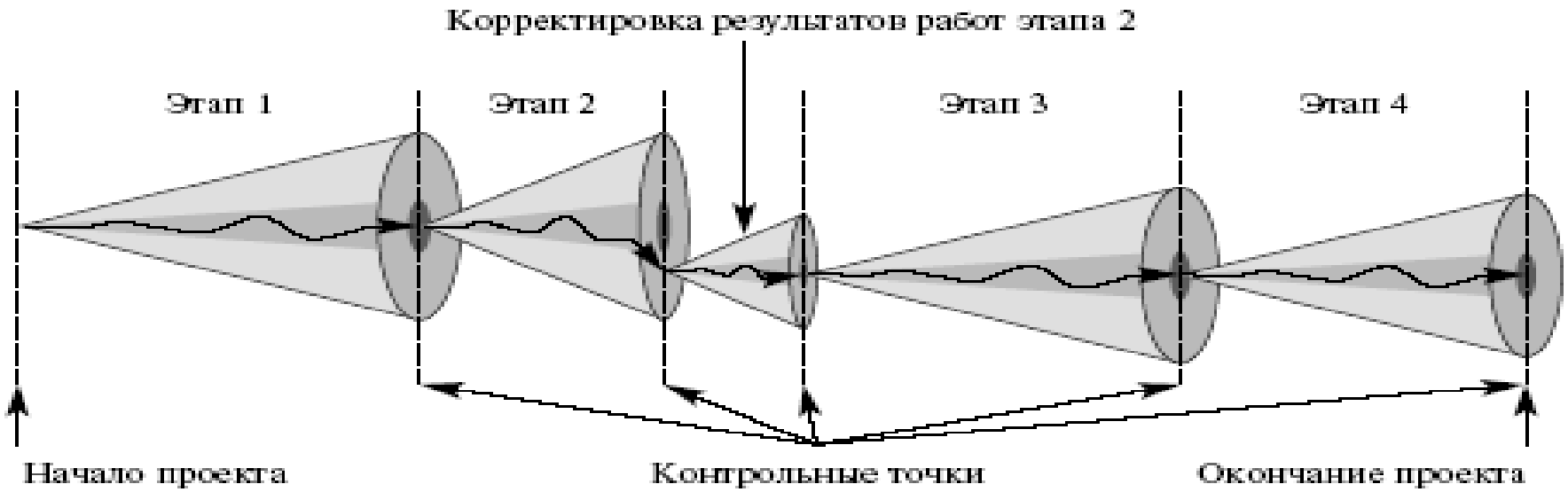
Знание моделей жизненного цикла помогает понять даже непрофессионалу, на что можно рассчитывать при заказе или приобретении программного обеспечения, а что нереально требовать от него

Конус операционных маршрутов проекта по разработке ПО



Одна из основных задач менеджмента программного проекта заключается в том, чтобы не допустить отклонения траекторий деятельности исполнителей от целевой области проекта. Для этого должны быть предусмотрены средства, позволяющие выявлять отклонения, и инструменты воздействия, предназначенные для их корректировки

Последовательное развитие проекта



Процесс разработки программного обеспечения разбивается на этапы. Каждый этап характеризуется:

- субъектом-исполнителем
- сроками, когда должны быть решены задачи
- выделенными ресурсами
- средствами, инструментами и методами решения задач
- контрольными мероприятиями, позволяющими удостовериться, что задачи этапа решены

Последовательные модели жизненного цикла

- Каскадная
- V-образная
- Упрощенный процесс системного проектирования

Краткое описание подхода к разработке ПО: Вначале собираются и анализируются требования. После их документирования и утверждения последовательно выполняются проектирование, кодирование и тестирование компонентов программной системы, после чего происходит их интеграция

После разработки документации система передается на сопровождение в условиях эксплуатации

Каждый этап выполняется однократно, возвратов на предыдущие этапы не предусматривается. Иногда отдельные этапы (например, связанные с верификацией) выполняются параллельно (V-образная модель)

Достоинства последовательных моделей ЖЦ

- Последовательные модели просты и удобны в использовании, так как процесс разработки выполняется поэтапно
- Они хорошо известны потребителям
- Модели хорошо срабатывают тогда, когда требования к качеству доминируют над требованиями к затратам и графику выполнения проекта
- Они способствуют осуществлению четкого менеджмента проекта, так как допускают использование инструментов временного планирования и контроля, поскольку момент завершения каждой фазы может рассматриваться в качестве контрольной точки, позволяющей проанализировать достигнутые результаты

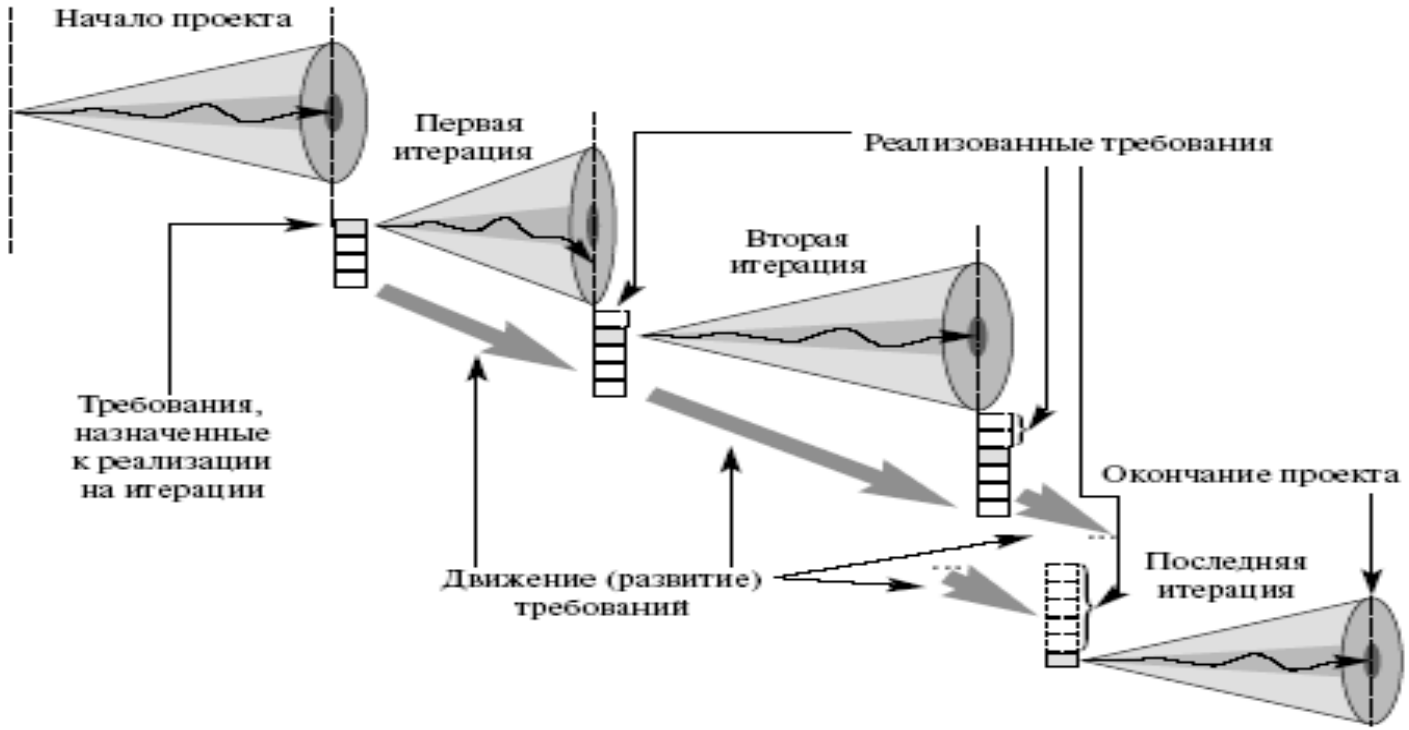
Недостатки последовательных моделей ЖЦ

- В основе разработки программной системы лежат спецификации, вследствие чего высок риск создания системы, не удовлетворяющей потребностям пользователей, так как пользователи не в состоянии сразу изложить все свои требования и/или за время разработки могут произойти изменения во внешней среде, которые повлияют на требования к системе
- В последовательных моделях не учтены итерации между фазами и не предусмотрено внесение динамических изменений на разных этапах жизненного цикла
- Интеграция является конечной, а не пошаговой операцией
- Согласование получаемых результатов с пользователями производится только в точках, планируемых после завершения каждой стадии
- В последовательные модели не входят действия, направленные на анализ рисков

Сферы применения последовательных моделей ЖЦ

- в случаях, когда информация о требованиях известна заранее и они стабильны в течение всего периода разработки
- в проектах с участием больших команд разработчиков
- в критически важных системах реального времени с высокими требованиями по надежности
- при разработке новой версии уже существующего продукта или переносе его на новую платформу
- в организациях, имеющих опыт создания программных систем определенного типа (например, бухгалтерский учет, начисление зарплаты и пр.)

Итеративное наращивание возможностей системы



На каждой итерации строится программное изделие, которое в принципе может применяться на практике, пусть даже не до конца реализуя функциональность в целом.

Рост количества прямоугольников при переходе от итерации к итерации отражает поступление новых требований в ходе развития проекта

Примечание: пунктиром выделены отработанные требования

Причины использования эволюционных и инкрементных моделей жизненного цикла ПО

«Самой тяжелой составляющей процесса построения программной системы является принятие однозначного решения о том, что именно необходимо построить. Ни одна из остальных составляющих работы над концепцией не представляет собой такую трудность, как определение детальных технических требований, включая все аспекты соприкосновения продукта с людьми, машинами и другими программными системами. Ни одна другая составляющая работы в такой степени не наносит ущерб полученной в результате системе, если она выполнена неправильно. Именно эту составляющую процесса разработки тяжелее всего исправить на более поздних этапах.

Следовательно, самая важная функция, которую выполняет разработчик клиентских программ, заключается в итеративном извлечении и уточнении требований к продукту. Ведь на самом деле клиент не имеет представления о том, что именно он хочет получить»

Ф. Брукс

Основные задачи, решаемые за счет создания прототипов:

- итеративное извлечение и уточнение требований к продукту
- возможность изменить эксплуатационное окружение программной системы
- снижение неопределенности (а следовательно, и рисков) по мере завершения каждой итерации

Эволюционные и инкрементные модели жизненного цикла

- Эволюционная модель быстрого прототипирования
- Модель быстрой разработки приложений RAD (Rapid Application Development)
- Инкрементная модель

Краткое описание подхода к разработке ПО: модели основываются на создании последовательности релизов и/или прототипов, критический анализ которых производится заказчиком. В процессе такого анализа уточняются требования к программному продукту

Данные модели чаще всего применяются в системах с ярко выраженным пользовательским интерфейсом

Достоинства эволюционных и инкрементных моделей ЖЦ

- Взаимодействие пользователя и/или заказчика с системой начинается на раннем этапе разработки, что минимизирует возможность их неудовлетворенности конечным продуктом и гарантирует соответствие программной системы коммерческим потребностям
- Благодаря раннему знакомству пользователя с системой сводится к минимуму количество неточностей в требованиях
- Модели позволяют выполнить несколько итераций на всех фазах жизненного цикла
- При использовании моделей заказчикам демонстрируются постоянные, видимые признаки прогресса в реализации проекта, что дает им возможность чувствовать себя более уверенно
- Благодаря меньшему объему доработок уменьшаются совокупные затраты на разработку и обеспечивается управление рисками
- Документация сконцентрирована на конечном продукте, а не на процессе его разработки

Недостатки итерационных моделей ЖЦ

- При использовании эволюционных прототипных моделей решение трудных проблем может отодвигаться на будущее. В результате конечный продукт может не оправдать тех надежд, которые возлагались на прототип
- На очередном итерационном этапе прототип представляет собой частичную реализацию системы, поэтому если выполнение проекта завершается досрочно, у конечного пользователя останется только лишь незавершенный продукт
- С учетом создания рабочих прототипов, качеству всего ПО или долгосрочной эксплуатационной надежности может быть уделено недостаточно внимания
- Разработанные прототипы часто страдают от неадекватной или недостающей документации
- Сложность завершения работы над проектом

Факторы, влияющие на выбор модели жизненного цикла

- ***характеристики требований***:
  - являются ли требования легко определимыми и стабильными в процессе разработки
  - требуется ли проверка концепции для демонстрации возможностей ПП
- ***характеристики участников команды разработчиков***:
  - являются ли проблемы предметной области, а также инструменты, используемые в проекте новыми для большинства разработчиков
  - изменяются ли роли участников проекта во время жизненного цикла
- ***характеристики коллектива пользователей***:
  - будет ли присутствие пользователей ограничено в жизненном цикле
  - будет ли заказчик отслеживать ход выполнения проекта
- ***характеристики типа проектов и рисков***:
  - будет ли проект идентифицировать новое направление продукта для организации
  - являются ли «прозрачными» интерфейсные модули
  - должна ли обеспечиваться высокая степень надежности

Качество программного обеспечения

- Определение ISO: Качество - это полнота свойств и характеристик продукта, процесса или услуги, которые обеспечивают способность удовлетворять заявленным или подразумеваемым потребностям
- Определение IEEE: Качество программного обеспечения - это степень, в которой оно обладает требуемой комбинацией свойств

Стандарты качества ПО:

- ISO/IEC 9126:1-4:2002 (ГОСТ Р ИСО/МЭК 9126-93)
- ISO/IEC 14598 – набор стандартов, регламентирующий способы оценки характеристик качества

В совокупности они образуют модель качества - SQuaRE (Software Quality Requirements and Evaluation)

Качество программного обеспечения

- внешнее качество, заданное требованиями заказчика в спецификациях и отражающееся характеристиками конечного продукта;
- внутреннее качество, проявляющееся в процессе разработки и других промежуточных этапах жизненного цикла ПС;
- качество при использовании в процессе нормальной эксплуатации и результативность достижения потребностей пользователей с учетом затрат ресурсов (эксплуатационное качество)

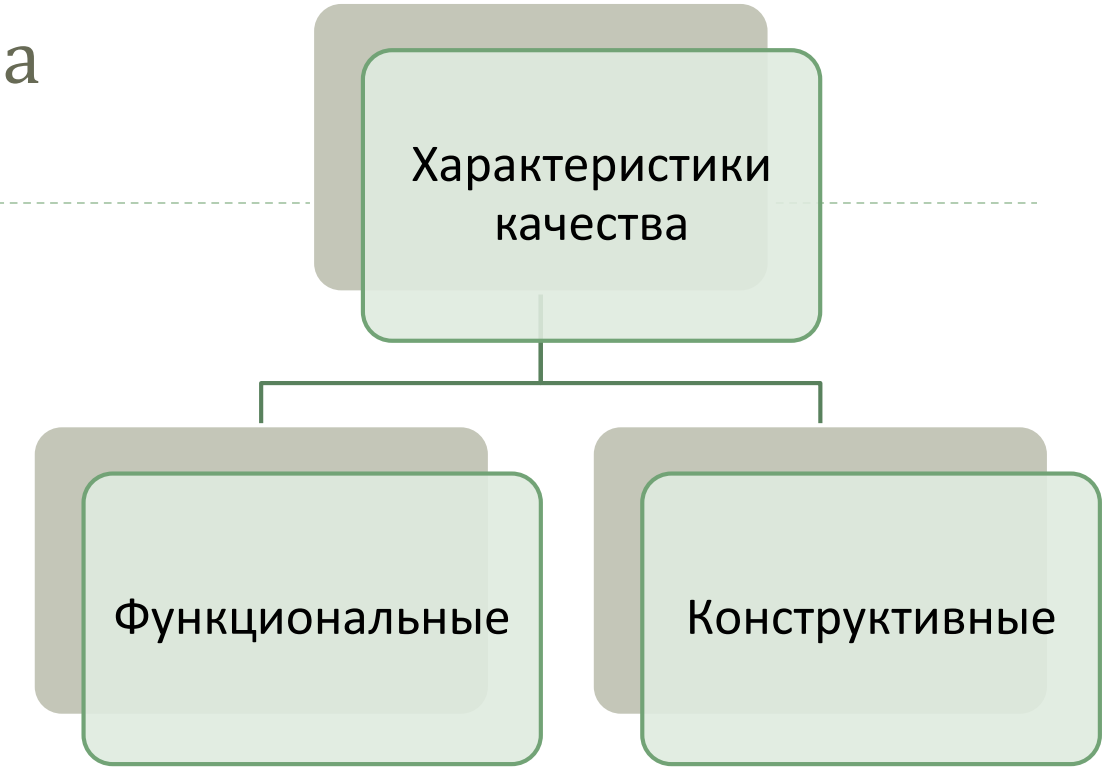
Качество объекта зависит от того, для какой **цели**, для какого **потребителя** и для каких **условий** делается его оценка

Стандарт ISO/IEC 9126:1-4: Информационная технология - Качество программных средств

- Часть 1: Модель качества
- Часть 2: Внешние метрики качества
- Часть 3: Внутренние метрики качества
- Часть 4: Метрики качества в использовании

6 базовых характеристик качества, используемых в модели стандарта ISO/IEC 9126-1	
функциональная пригодность	практичность
надежность	сопровождаемость
эффективность	мобильность

Характеристики и метрики качества ПО



Метрики качества:

- категорийные, или описательные (номинальные) метрики используются для оценки функциональных возможностей программных средств;
- количественные метрики применимы для измерения надежности и эффективности сложных комплексов программ;
- качественные метрики в наибольшей степени соответствуют практичности, сопровождаемости и мобильности программных средств

- определяют назначение ПО и задачи, которые оно решает для основных пользователей
- отличаются широким разнообразием
- трудно поддаются унификации
- классифицируются в зависимости от свойств и назначения

- имеют небольшую номенклатуру, которая может быть унифицирована
- могут быть адаптированы и использованы для описания остальных внутренних и внешних характеристик качества, закрепленных в стандарте

Набор стандартов ISO 9000

- ISO 9000:2000 Quality management systems – Fundamentals and vocabulary. Системы управления качеством – Основы и словарь

- ISO 9001:2000 Quality management systems – Requirements. Models for quality assurance in design, development, production, installation and servicing. Системы управления качеством – Требования. Модели для обеспечения качества при проектировании, разработке, производстве, установке и обслуживании. Определяет общие правила обеспечения качества результатов во всех процессах жизненного цикла

"ГОСТ Р 57189-2016/ISO/TS 9002:2016. Национальный стандарт Российской Федерации. Системы менеджмента качества. Руководство по применению ИСО 9001:2015 (ISO/TS 9002:2016, IDT)" (утв. Приказом Росстандарта от 25.10.2016 N 1499-ст)

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ

И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 57189-2016/ISO/TS 9002:2016

СИСТЕМЫ МЕНЕДЖМЕНТА КАЧЕСТВА

РУКОВОДСТВО ПО ПРИМЕНЕНИЮ ИСО 9001:2015

(ISO/TS 9002:2016, IDT)

Набор стандартов ISO 9000 регулирует общие принципы обеспечения качества процессов производства во всех отраслях экономики

Capability Maturity Model — модель CMM — модель зрелости возможностей создания программного обеспечения

- Зрелость процессов разработки ПО — это степень их управляемости, контролируемости и эффективности
- Для этого требуется документирование процессов и доведение этих документов до сведения персонала организации
- Не менее важен постоянный контроль за ходом выполнения процессов и их постоянное совершенствование
- В организациях, достигших технологической зрелости, процессы создания и сопровождения ПО принимают статус стандарта, фиксируются в организационных структурах и определяют производственную тактику и стратегию. Как следствие, возникает необходимость построения необходимой инфраструктуры и создания требуемой корпоративной культуры производства

Чем выше уровень зрелости, тем более предсказуемыми и управляемыми становятся процессы и, как следствие, более предсказуемо и качественно будут реализованы проекты

Примечание: Разработчик модели – американский институт Software Engineering Institute (SEI), Заказчик – Федеральное правительство США

Модель CMM (5 уровней зрелости организации, занимающейся разработкой ПО)

Начальный. Самый примитивный статус организации. Организация способна разрабатывать ПО, но при этом не имеет явно осознанного процесса, и качество продукта целиком определяется индивидуальными способностями разработчиков. Команда следует указаниям того, кто проявляет инициативу. Успех одного проекта не гарантирует успех другого. При завершении проекта не фиксируются данные о трудозатратах, расписании и качестве

Повторяемый. В некоторой степени отслеживается процесс. Делаются записи о трудозатратах и планах. Функциональность каждого проекта описана в письменной форме

Установленный. Имеют определённый, документированный и установленный процесс работы, не зависящий от отдельных личностей. Вводятся согласованные профессиональные стандарты, а разработчики их выполняют. Такие организации в состоянии достаточно надёжно предсказывать затраты на проекты, аналогичные выполненным ранее

Управляемый. Могут точно предсказать сроки и стоимость работ. Есть база данных накопленных измерений, но нет изменений при появлении новых технологий и парадигм

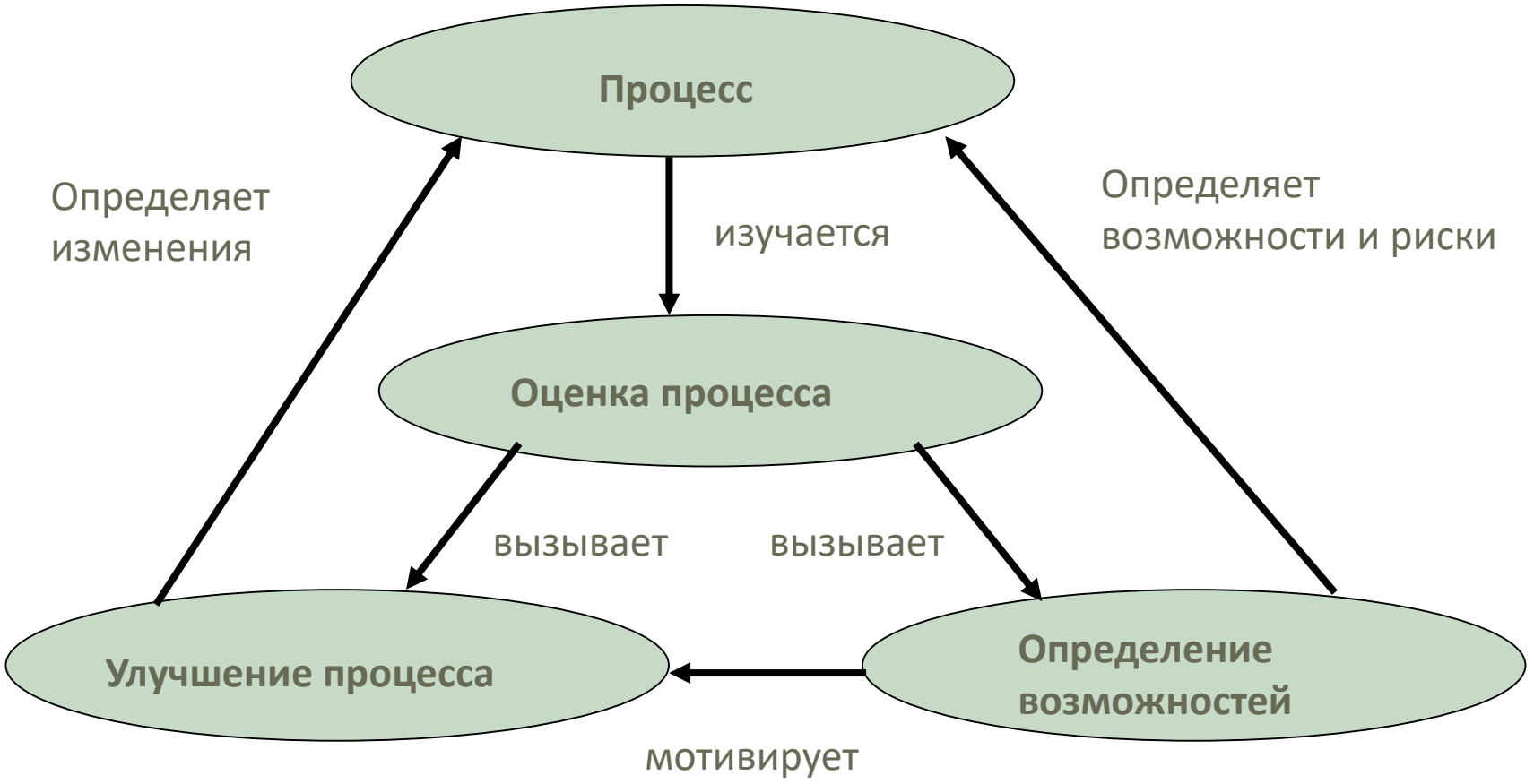
Оптимизированный. Есть постоянно действующая процедура поиска и освоения новых улучшенных методов и инструментов

Особенности совершенствования процессов при переходе с уровня на уровень

Ограничения на использование модели CMM:

- стандарт CMM является собственностью Software Engineering Institute и не является общедоступным (в частности, дальнейшая разработка стандарта ведется самим институтом, без заметного влияния остальной части программистского сообщества);
- оценка качества процессов организаций может проводиться только специалистами, прошедшими специальное обучение и аккредитованными SEI;
- стандарт ориентирован на применение в относительно крупных компаниях

Основные элементы стандарта ISO/IEC 15504 (SPICE)



- ***Оценка процесса*** происходит путем сравнения процесса разработки ПО, существующего в данной организации, с описанной в стандарте моделью.
- ***Определение возможностей*** процесса происходит на основе выбора одного из альтернативных путей перехода к более совершенному процессу
- ***Улучшение*** того или иного ***процесса*** происходит а основе сформулированных на предыдущих этапах целей и направлено на реализацию выбранной стратегии перехода

Модель Гантера: модель фазы-функции

Модель жизненного цикла программного обеспечения может использоваться для комплексного управления программным проектом, если наложить на нее контрольные точки, задающие организационно-временные рамки проекта, и организационно-технические, так называемые производственные функции, которые выполняются при его развитии

Этот подход реализован в модели Гантера в виде матрицы «фазы — функции»

Модель Гантера имеет два измерения:

- фазовое, отражающее этапы выполнения проекта и сопутствующие им события
- функциональное, показывающее, какие производственные функции выполняются в ходе развития проекта, и какова их интенсивность на каждом из этапов

Модель фазы-функции (фазовое измерение)

Основные этапы и контрольные точки

- *Этап исследования* — для проекта обосновываются необходимые ресурсы (контрольная точка 1) и формулируются требования к разрабатываемому изделию (контрольная точка 2)
- *Анализ осуществимости* —определяется возможность конструирования программного средства с технической точки зрения (достаточно ли ресурсов, квалификации персонала и т.п.), решаются вопросы экономической и коммерческой эффективности
- *Конструирование* — начинается обычно на этапе анализа осуществимости, как только документально зафиксированы предварительные цели проекта (контрольная точка 2), и заканчивается утверждением проектных решений в виде официальной спецификации на разработку (контрольная точка 5)
- *Программирование* — программирование компонентов с последующей сборкой системы. Этап завершается, когда разработчики заканчивают документирование, отладку и компоновку и передают ПС службе, выполняющей независимую оценку результатов работы (независимые испытания начались — контрольная точка 7)
- *Оценка* — является буферной зоной между началом испытаний и практическим использованием изделия. Этап начинается, как только проведены внутренние (силами разработчиков) испытания изделия (контрольная точка 6) и заканчивается, когда подтверждается готовность изделия к эксплуатации (контрольная точка 9)
- *Использование* — начинается в момент передачи изделия на распространение (контрольная точка 8). Этап продолжается, пока изделие интенсивно эксплуатируется. Он связан с внедрением, обучением, настройкой и сопровождением, возможно, с модернизацией изделия. Этап заканчивается, когда разработчики прекращают систематическую деятельность по сопровождению и поддержке данного программного изделия (контрольная точка 10)

Модель фазы – функции (функциональное измерение)

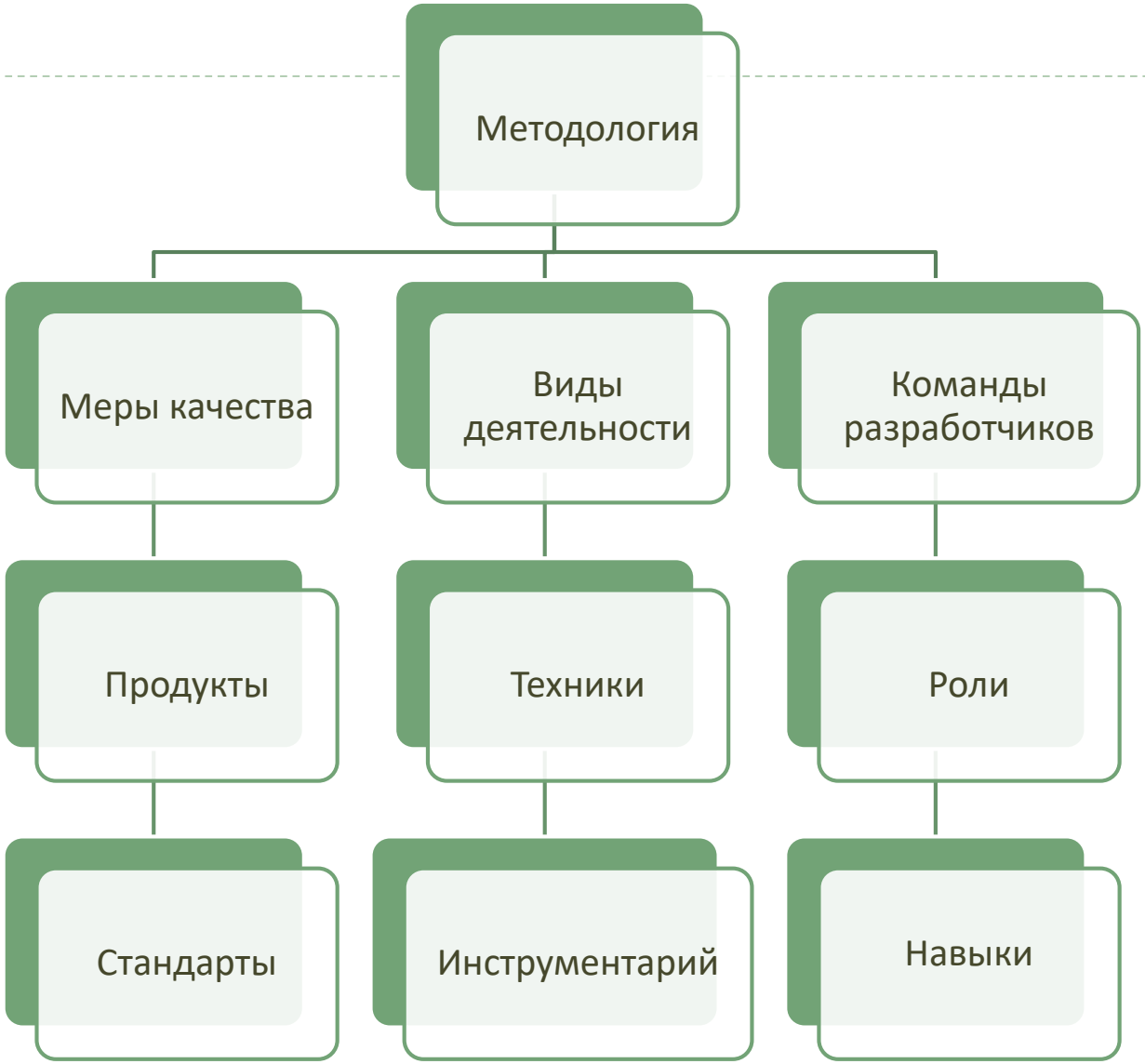
Основные функции

- *Планирование* — содержание того, что должно планироваться на каждом из этапов определяется по характеру контрольных точек. Конкретные методы планирования определяются концепциями, принимаемыми для данного проекта
- *Разработка* – эта функция пронизывает весь проект и каждый раз направлена на получение рабочего продукта этапа
- *Обслуживание* — функция, обеспечивающая максимально комфортную обстановку для выполнения некоторой деятельности. По интенсивности обслуживания можно косвенно судить о качестве организации работ проекта
- *Выпуск документации*. Документация в проекте рассматривается как полноправный вид рабочих продуктов, сопровождающий другие рабочие продукты: программы, диаграммы моделей системы и прочее
- *Испытания*. Испытывать приходится все рабочие продукты проекта
- *Поддержка* использования рабочих продуктов — это функция, выполнение которой необходимо в связи с передачей продукта в эксплуатацию. Содержание ее связано с обучением и созданием необходимой для использования продуктов инфраструктуры
- *Сопровождение* (по Гантеру) отличается от поддержки тем, что оно организуется для внешнего использования продуктов

Методологии программирования

Методология – ряд связанных между собой методов или техник

Толковый словарь Webster



Методологии 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

Rational Unified Process (RUP): три измерения

- Rational Unified Process — это **процесс** разработки программного обеспечения, который обеспечивает упорядоченный подход и распределение обязанностей в организации – разработчике. Целью этого процесса является производство качественного ПО, удовлетворяющего требованиям конечных пользователей, в рамках прогнозируемого бюджета и графика работ
- Rational Unified Process — это **продукт процесса**, разработанный корпорацией Rational Software (вошла в состав IBM в 2003 г.). Он неразрывно связан с выпускаемым корпорацией набором средств для разработки программного обеспечения
- Rational Unified Process — это еще и **контур процесса**, который можно адаптировать и расширить для удовлетворения требований принявшей его организации

На вопрос «Что такое RUP» есть, по крайней мере, три корректных ответа:
RUP — философия и практика успешной разработки программного обеспечения
RUP — формальное описание процесса
RUP — готовый сайт, содержащий формальное описание процесса

6 базовых принципов RUP	
Итеративная разработка	Использование визуального моделирования
Управление требованиями	Постоянный контроль качества
Использование модульных архитектур	Отслеживание изменений

Rational Unified Process обеспечивает производство качественных систем с помощью процесса, который можно повторить, а результаты действия которого можно предсказать

RUP как процесс разработки программного обеспечения

Процесс – частично упорядоченный набор шагов, который надо проделать для достижения цели; при разработке ПО цель состоит в создании или расширении существующего программного изделия. Разработка и все последующие доработки составляют жизненный цикл программной системы

RUP - это итеративный процесс, который обеспечивает большую гибкость при учете новых требований и позволяет проекту заранее идентифицировать и разрешать риски	RUP – это управляемый процесс, который предполагает управление требованиями и управление изменениями
RUP – это процесс, основанный на моделях – семантически богатых представлениях программной системы при ее разработке	RUP уделяет большое внимание первоначальной разработке устойчивой архитектуры программы, которая минимизирует переделки, обеспечивает возможность повторного использования кода и надежность эксплуатации
RUP поддерживает объектно-ориентированную технологию, при этом Унифицированный язык моделирования (UML) используется в качестве общей системы обозначений	RUP поощряет управление качеством, используя для этого объективные измерения и критерии

RUP как продукт процесса разработки программного обеспечения

«Процесс разработки программного обеспечения — это тоже программное обеспечение»
Ли Остервейл

Продукт Rational Unified Process входит в состав Rational Suite всех комплектаций и включает в себя:

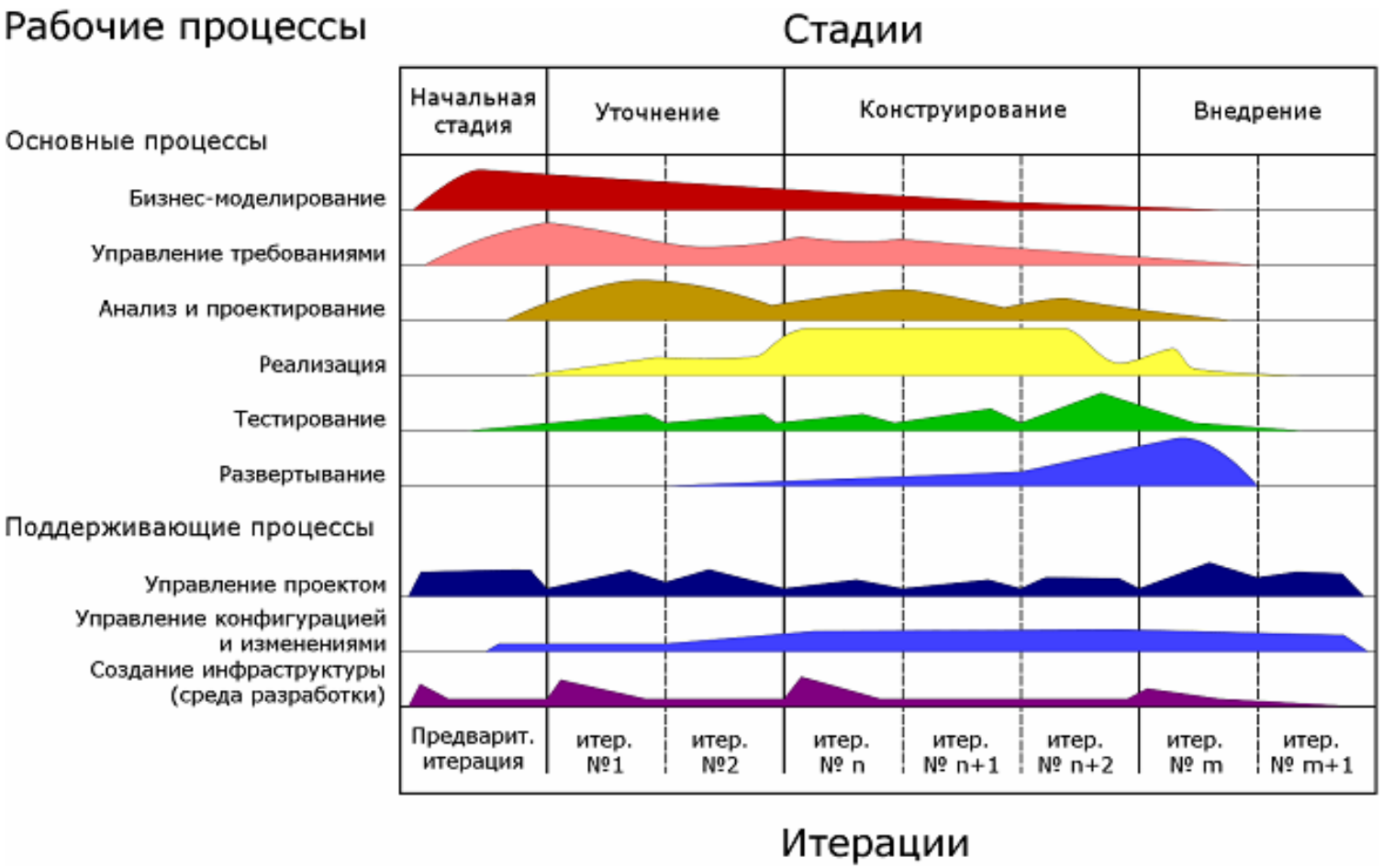
- интерактивную версию базы знаний в формате html;
- комплект документов, описывающих процесс

База знаний содержит описание архитектуры процесса и его составляющих (стадий разработки и потоков работ), технологии работы, правил создания различных артефактов (моделей, отчетов, планов и пр.) и порядка использования средств инструментальной поддержки
В состав пакета входят настраиваемые шаблоны, которые позволяют использовать информацию, порожденную в ходе работы над проектом, при создании отчетов в приложениях Microsoft Word, Microsoft Project и др.

Rational Unified Process проектируется, разрабатывается, сдается и эксплуатируется подобно любому программному продукту:

- регулярно выпускаются обновленные версии Rational Unified Process
- продукт предоставляется оперативно, с использованием технологии Web, поэтому для разработчиков он буквально доступен с клавиатуры
- Он составляет единое целое с множеством средств разработки программного обеспечения, поэтому для каждого элемента процесса предлагается подробное руководство разработчика

Модель жизненного цикла RUP



RUP организует работу над проектом в терминах последовательности действий (workflows), продуктов деятельности, исполнителей и других статических аспектов процесса с одной стороны, и в терминах циклов, фаз, итераций и временных отметок завершения определенных этапов в создании ПО (milestones), т.е. в терминах динамических аспектов процесса, с другой

- Проект в RUP состоит из последовательности итераций с рекомендованной продолжительностью от 2 до 6 недель
- Основной единицей планирования итераций является прецедент использования, который представляет собой описание сценария взаимодействия пользователя с системой, полностью выполняющего конкретную пользовательскую задачу
- Перед началом очередной итерации определяется набор прецедентов использования, которые будут реализованы к ее завершению
- Итеративная модель позволяет вносить необходимые изменения в требования, проектные решения и реализацию в ходе проекта

Фазы модели RUP

- **Фаза начала проекта (Inception).** Определяются основные цели проекта, его бюджет, а также основные средства его выполнения — технологии, инструменты, ключевой персонал, составляются предварительные планы проекта. Основная цель этой фазы — достичь компромисса между всеми заинтересованными лицами относительно задач проекта
- **Фаза уточнения (Elaboration).** Основная цель этой фазы — на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы
- **Фаза конструирования (Construction).** На данной фазе происходит детальное прояснение требований и разработка системы, удовлетворяющей этим требованиям, на основе спроектированной ранее архитектуры
- **Фаза внедрения (Transition).** Цель фазы — сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей

Технологические процессы

- Бизнес моделирование: исследование и описание существующих бизнес-процессов заказчика, а также поиск их возможных улучшений
- Управление требованиями: определение границ проекта, разработка функционального дизайна будущей системы и его согласование с заказчиком
- Анализ и проектирование: проектирование архитектуры системы на основе функциональных требований и ее развитие на протяжении всего проекта
- Реализация: разработка, модульное тестирование и интеграция компонентов системы
- Тестирование: поиск и отслеживание дефектов в системе, проверка корректности реализации требований
- Развертывание: создание дистрибутива, установка системы, обучение пользователей
- Управление проектом: создание проектной команды, планирование фаз и итераций, управление бюджетом и рисками
- Управление конфигурацией и требованиями: управление версиями исходного кода и документации, процесс обработки запросов на изменение (change requests)
- Создание инфраструктуры: создание инфраструктуры для выполнения проекта, включая организацию и настройку процесса разработки

Набор инструментов для поддержки RUP

- **Rational Requisite Pro** - поддерживает обновления и отслеживает изменения в требованиях для всего коллектива разработчиков, представляя их в удобном виде для чтения, обсуждения и изменений
- **Rational ClearQuest** - Windows и Web-размещаемый продукт, который помогает коллективу разработчиков отслеживать и управлять всеми действиями по изменению ПО в течение его жизненного цикла
- **Rational Rose** - средство визуального моделирования для бизнес процессов, анализа требований, и проектирования на основе архитектуры компонентов
- **Rational SoDa**- автоматизирует создание документации для всего процесса разработки ПО, значительно сокращая стоимость документации и время на ее создание
- **Rational Purify** - средство поиска ошибок на run-time для разработчиков приложений и компонентов, программирующих на C/C++; помогает находить ошибки утечки памяти
- **Rational Visual Quantify** - средство измерения характеристик для разработчиков приложений и компонентов, программирующих на C/C++, Visual Basic и Java; помогает определять и устранять узкие места в производительности ПО
- **Rational Visual PureCoverage** - автоматически определяет области кода, которые не подвергаются тестированию; разработчики могут учесть это и более тщательно выполнять проверку
- **SQA TeamTest** - создает, обслуживает и выполняет автоматизированные функциональные тесты, позволяя тщательно протестировать код и проверить, соответствует ли ПО предъявляемым к нему требованиям
- **Rational PerformanceStudio** - простое в использовании, точное и масштабируемое средство, которое измеряет и предсказывает характеристики клиент/серверных и Web систем
- **Rational ClearCase** - лидирующее на рынке средство конфигурационного управления, позволяющее менеджерам проекта отслеживать эволюцию каждого разрабатываемого проекта

Общая логика модели RUP

Что разрабатывается?	• артефакт – любая значимая внутренняя или подлежащая сдаче порция информации, которая играет определенную роль в разработке системы
Кто и как разрабатывает?	• исполнители – описание обязанностей физических лиц и того, как они должны действовать • вид деятельности – задача, которую выполняет исполнитель, чтобы создать артефакт
Чем пользуются исполнители?	• шаблоны - модели (или прототипы) артефактов • инструментальные наставники
Чем руководствуются исполнители?	• директивы - правила, рекомендации или эвристические советы, поддерживающие виды деятельности и их этапы

Кто использует Rational Unified Process?

- Телекоммуникация: Ericsson, Alcatel, MCI
- Транспорт, авиационно-космическая отрасль, оборонная промышленность: Lockheed-Martin, British Aerospace
- Промышленность: Xerox, Volvo, Intel
- Финансы: Visa, Merrill Lynch, Schwab
- Интеграторы систем: Ernst & Young, Oracle, Deloitte & Touche

Типовой жизненный цикл методологии быстрого развития

- Начальная фаза
- Серия максимально коротких итераций, состоящих из следующих 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 проекта

Затраты являются наиболее ограничивающей переменной: вы не можете напрямую менять деньги на качество, объем работ или скорость, с которой выпускаются промежуточные версии системы. В начале проекта вообще не существует возможности потратить слишком много денег, поэтому инвестирование надо выполнять понемногу, увеличивая с течением времени объемы вложений

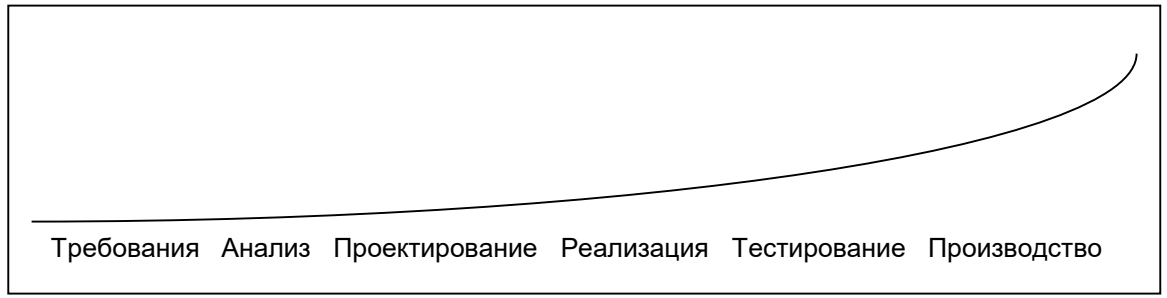
Экстремальное программирование

- все задание разбивается на формулировки подзадач, каждая формулировка должна иметь срок выполнения от 1 до 3-х недель, если больше, то – следует разделить, меньше – объединить
- путем равновесия объема, времени, ресурсов и качества составляется структурный план выпуска версий программы на основании пользовательских формулировок
- перед началом новой итерации создается ее подробный план: формулировки разбиваются па задачи сроком от 1 до 3-х дней и сортируются по важности, добавляются задачи, которые не смогли пройти тест приемки
- проводятся ежедневные утренние планерки, желательно стоя, чтобы не занимали слишком много времени
- приветствуется простой дизайн проекта
- создаваемым классам даются простые имена
- осуществляется постоянный контакт с заказчиком

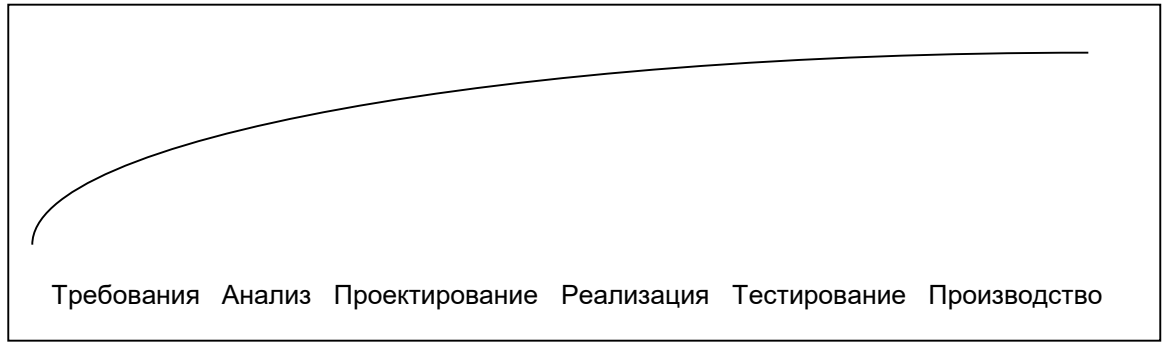
Экстремальное программирование применяется для создания программ коллективом от 2-х до 10 программистов при заранее заданных сроках сдачи проекта

Стоимость внесения изменений в ПО по мере разработки

раньше



сейчас



Одно из основных предположений экстремального программирования: если стоимость внесения изменений растет медленно, вы можете откладывать решение важных задач на более поздние сроки. Сегодня вы должны реализовать лишь то, без чего сегодня не обойтись

- **Коммуникации**: учитывая, что XP-проекты не опираются на документацию, большую роль играет живое общение в командах. Поэтому в таких проектах всегда тщательно следят за тем, чтобы люди общались, когда это нужно
- **Простота**: требование простоты относится и к дизайну системы, и к программному коду, которые должны разрабатываться настолько просто, насколько это возможно в текущей ситуации. Подход XP: лучше сделать простую вещь сегодня и затратить чуть больше усилий завтра, для того, чтобы модифицировать ее, если в этом возникнет необходимость
- **Обратная связь**: самым лучшим способом обратной связи является эксплуатация системы в реальных производственных условиях при одновременном ее развитии за счет добавления новой функциональности (обратная связь в масштабе недель и месяцев). Обратная связь в масштабе минут и дней обеспечивается написанием тестов
- **Храбрость**: в контексте первых трех ценностей необходимо действовать с максимально возможной скоростью. Для этого нужна храбрость. Прежде всего это касается отказа от ранее разработанного кода

Жизненный цикл XP проекта

- Жизненный цикл проекта в XP состоит из последовательности релизов
- Каждый релиз – это полноценная версия продукта, которую может использовать заказчик, содержащая дополнительную функциональность по сравнению с предыдущим релизом
- Релиз создается в результате одной или нескольких итераций, длящихся от одной до четырех недель

Шаги планирования релиза:

- Заказчик формулирует свои требования в виде историй, которые оцениваются по трудоемкости разработчиками. Оценки трудоемкости делаются в так называемых идеальных днях – времени, которое некий воображаемый разработчик потратит на реализацию истории при полном отсутствии отвлекающих факторов, параллельных задач, перерывов и т.п.
- Истории сортируются по приоритету, рискам и сложности реализации
- Определяется фактическая производительность команды (какое число реальных дней соответствует идеальному дню)
- На основании фактической производительности определяется, какие истории войдут в очередной релиз и за какое время он будет завершен

Основные методики XP

- **Игра в планирование** (planning game) — быстро определяет перечень задач (объем работ), которые необходимо реализовать в следующей версии продукта, для этого рассматриваются бизнес-приоритеты и технические оценки
- **Небольшие версии** (small releases) — самая первая упрощенная версия системы быстро вводится в эксплуатацию, после этого через относительно короткие промежутки времени происходит выпуск версии за версией
- **Метафора** (metaphor) — эта простая общедоступная и общеизвестная история, которая коротко описывает, как работает вся система. Эта история управляет всем процессом разработки
- **Простой дизайн** (simple design) — в каждый момент времени система должна быть спроектирована так просто, как это возможно. Чрезмерная сложность устраняется, как только ее обнаруживают
- **Тестирование** (testing) — программисты постоянно пишут тесты для модулей. Заказчики пишут тесты, которые демонстрируют работоспособность и завершенность той или иной возможности системы
- **Переработка кода** (refactoring) — программисты реструктурируют систему, не изменяя при этом ее поведения. При этом они устраняют дублирование кода, улучшают коммуникацию, упрощают код и повышают его гибкость.
- **Программирование парами** (pair programming) — весь разрабатываемый код пишется двумя программистами на одном компьютере
- **Коллективное владение кодом** (collective ownership) — в любой момент времени любой член команды может изменить любой код в любом месте системы
- **Непрерывная интеграция** (continuous integration) — система интегрируется и собирается множество раз в день. Это происходит каждый раз, когда завершается решение очередной задачи
- **40-часовая_неделя** (40-hour week) — программисты работают не более 40 часов в неделю. Никогда нельзя работать сверхурочно две недели подряд
- **Заказчик па месте разработки** (on-site customer) — в состав команды входит реальный живой пользователь системы. Он доступен в течение всего рабочего дня и способен отвечать на вопросы о системе
- **Стандарты кодирования** (coding standards) — программисты пишут весь код в соответствии с правилами, которые обеспечивают коммуникацию при помощи кода

Пример гибкой методологии: SCRUM

Источник:
https://m.youtube.com/watch?time_continue=212&v=yJgw8W65b5s&embeds_euri=https%3A%2F%2Fyastatic.net%2F&source_ve_path=Mjg2NjY&feature=emb_logo