Технология разработки программного обеспечения 2015-2016 гг.

Лекции 1-2: разработка ПО

Пудов Сергей Григорьевич

Немного статистики

- Известно, что 30 – 40 % проектов по разработке ПС не доходят до завершения.
- Около 70 % всех проектов реализуют поставленные задачи не полностью.
- Средний проект завершается с опозданием на 220%.
- В 10 % проектов результат не соответствует требованиям. В 12 % заказчик недостаточно привлекался к работе, чтобы обеспечить требуемые характеристики продукта. В 22 % проектов не все вносимые изменения принимались во внимание.

Немного статистики

Процессы жизненного цикла ПО

Процессы жизненного цикла ПО

ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств», идентичный международному стандарту ISO/IEC 12207:2008 «System and software engineering — Software life cycle processes»

Основные:

- ***Приобретение*** (действия и задачи заказчика, приобретающего ПО)
- ***Поставка*** (действия и задачи поставщика, который снабжает заказчика программным продуктом или услугой)
- ***Разработка*** (действия и задачи, выполняемые разработчиком: создание ПО, оформление проектной и эксплуатационной документации, подготовка тестовых и учебных материалов и т. д.)
- ***Эксплуатация*** (действия и задачи оператора — организации, эксплуатирующей систему)
- ***Сопровождение*** (действия и задачи, выполняемые сопровождающей организацией, то есть службой сопровождения). Сопровождение — внесений изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

Процессы жизненного цикла ПО

Вспомогательные

- ***Документирование*** (формализованное описание информации, созданной в течение ЖЦ ПО)
- ***Управление конфигурацией*** (применение административных и технических процедур на всем протяжении ЖЦ ПО для определения состояния компонентов ПО, управления его модификациями).
- ***Обеспечение качества*** (обеспечение гарантий того, что ИС и процессы ее ЖЦ соответствуют заданным требованиям и утвержденным планам)
- ***Верификация*** (определение того, что программные продукты, являющиеся результатами некоторого действия, полностью удовлетворяют требованиям или условиям, обусловленным предшествующими действиями)
- ***Аттестация*** (определение полноты соответствия заданных требований и созданной системы их конкретному функциональному назначению)
- ***Совместная оценка*** (оценка состояния работ по проекту: контроль планирования и управления ресурсами, персоналом, аппаратурой, инструментальными средствами)
- ***Аудит*** (определение соответствия требованиям, планам и условиям договора)
- ***Разрешение проблем*** (анализ и решение проблем, независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения или других процессов)

Процессы жизненного цикла ПО

Организационные

- ***Управление*** (действия и задачи, которые могут выполняться любой стороной, управляющей своими процессами)
- ***Создание инфраструктуры*** (выбор и сопровождение технологии, стандартов и инструментальных средств, выбор и установка аппаратных и программных средств, используемых для разработки, эксплуатации или сопровождения ПО)
- ***Усовершенствование*** (оценка, измерение, контроль и усовершенствование процессов ЖЦ)
- ***Обучение*** (первоначальное обучение и последующее постоянное повышение квалификации персонала)

Стадии жизненного цикла ПО

1. формирование требований к ПО;
2. проектирование (разработка системного проекта);
3. реализация (может быть разбита на подэтапы: детальное проектирование, кодирование);
4. тестирование (может быть разбито на автономное и комплексное тестирование и интеграцию);
5. ввод в действие (внедрение);
6. эксплуатация и сопровождение;
7. снятие с эксплуатации.

Базовые стратегии разработки ПО

Начальный этап:

кодирование – устранение ошибок

Три базовые стратегии:

- каскадная;
- инкрементная;
- эволюционная.

Определяются характеристиками:

- проекта;
- требований к продукту;
- команды разработчиков;
- команды пользователей.

Каскадная стратегия

Основными достоинствами каскадной стратегии, проявляемыми при разработке соответствующего ей проекта, являются:

1) **стабильность** требований в течение ЖЦ разработки;

2) необходимость только **одного прохода** этапов разработки, что обеспечивает простоту применения стратегии;

3) **простота** планирования, контроля и управления проектом;

4) доступность для **понимания** заказчиками.

Каскадная стратегия

К *основным* ***недостаткам*** *каскадной стратегии*, проявляемым при ее использовании в проекте, ей не соответствующем, следует отнести:

1) **сложность** полного формулирования требований в начале процесса разработки и невозможность их динамического изменения на протяжении ЖЦ;

2) **линейность** структуры процесса разработки; разрабатываемые ПС или системы обычно слишком велики и сложны, чтобы все работы по их созданию выполнять однократно; в результате возврат к предыдущим шагам для решения возникающих проблем приводит к увеличению финансовых затрат и нарушению графика работ;

3) **непригодность** промежуточных продуктов для использования;

4) **недостаточное участие пользователя** в процессе разработки ПС – только в самом начале (при разработке требований) и в конце (во время приемочных испытаний); это приводит к невозможности предварительной оценки пользователем качества программного средства или системы.

Каскадная стратегия

Области применения *каскадной стратегии* определяются ее достоинствами и ограничены ее недостатками. Использование данной стратегии наиболее эффективно в следующих случаях:

1) при разработке проектов с четкими, неизменяемыми в течение ЖЦ требованиями и понятной реализацией;

2) при разработке проектов невысокой сложности, например:

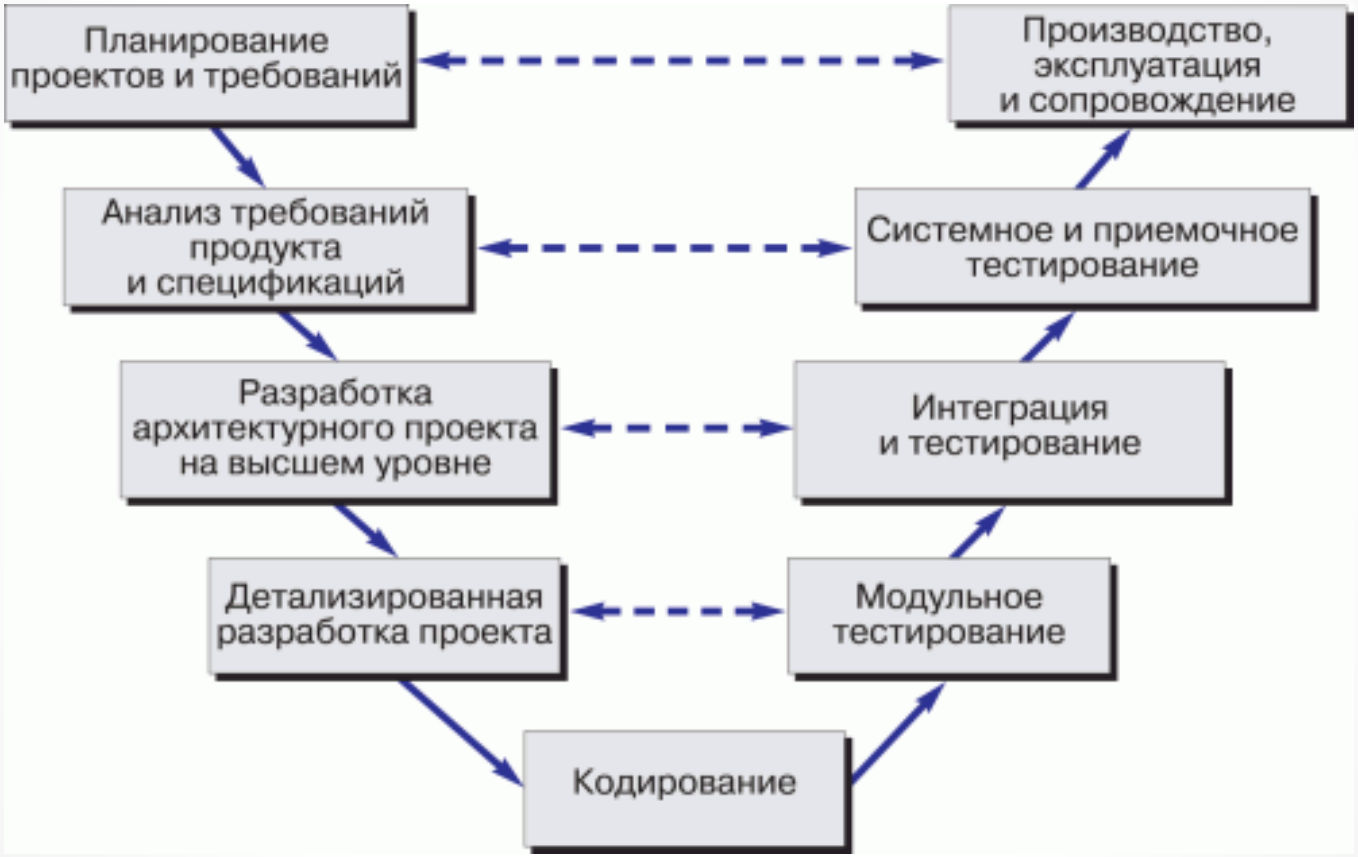
- создание программного средства или системы такого же типа, как уже разрабатывались разработчиками;
- создание новой версии уже существующего программного средства или системы;
- перенос уже существующего продукта на новую платформу;

3) при выполнении больших проектов в качестве составной части моделей ЖЦ, реализующих другие стратегии разработки

Каскадная модель (классический жизненный цикл)

Каскадная модель (с обратными связями)

V- модель


Планирование проектов и требований
Производство, эксплуатация и сопровождение
Анализ требований продукта и спецификаций
Системное и приемочное тестирование
Разработка архитектурного проекта на высшем уровне
Интеграция и тестирование
Детализированная разработка проекта
Модульное тестирование
Кодирование

Инкрементная стратегия

Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству (системе) в начале процесса разработки. Однако полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

Результат каждого цикла называется инкрементом.

Первый инкремент реализует базовые функции программного средства. В последующих инкрементах функции программного средства постепенно расширяются, пока не будет реализован весь набор требований. Различия между инкрементами соседних циклов в ходе разработки постепенно уменьшаются.

Результат каждого цикла разработки может распространяться в качестве
очередной поставляемой версии программного средства или системы.

Инкрементная стратегия

Достоинства:

1. возможность получения функционального **продукта** после реализации каждого инкремента;
2. **короткая продолжительность** создания инкремента; это приводит к сокращению сроков начальной поставки, позволяет снизить затраты на первоначальную и последующие поставки программного продукта;
3. предотвращение реализации громоздких спецификаций требований; стабильность требований во время создания определенного инкремента; возможность учета изменившихся требований;
4. **снижение рисков** по сравнению с каскадной стратегией;
5. **включение в процесс пользователей**, что позволяет оценить функциональные возможности продукта на более ранних этапах разработки и в конечном итоге приводит к повышению качества программного продукта, снижению затрат и времени на его разработку.

Инкрементная стратегия

Недостатки:

1. необходимость полного функционального определения системы или программного средства в начале ЖЦ для обеспечения планирования инкрементов и управления проектом;
2. возможность текущего **изменения требований** к системе или программному средству, которые уже реализованы в предыдущих инкрементах;
3. сложность планирования и распределения работ;
4. проявление **человеческого фактора**, связанного с тенденцией к оттягиванию решения трудных проблем на поздние инкременты, что может нарушить график работ или снизить качество программного продукта.

Инкрементная стратегия

Область применения

- при разработке проектов, в которых большинство требований можно сформулировать заранее, но часть из них могут быть уточнены через определенный период времени;
- при разработке сложных проектов с заранее сформулированными требованиями; для них разработка системы или программного средства за один цикл связана с большими трудностями;
- при необходимости быстро поставить на рынок продукт, имеющий базовые функциональные свойства;
- при разработке проектов с низкой или средней степенью рисков;
- при выполнении проекта с применением новых технологий

Инкрементные модели

- Классические вариант
- Модели с уточнением требований

Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными). План следующего инкремента предусматривает модификацию базового продукта, обеспечивающую дополнительные характеристики и функциональность.

Инкремент разрабатывается с помощью V-модели или спиральной модели

Эволюционная стратегия

Эволюционная стратегия представляет собой многократный проход этапов разработки. Данная стратегия основана на частичном определении требований к разрабатываемому программному средству или системе в начале процесса разработки. Требования постепенно уточняются в последовательных циклах разработки. Результат каждого цикла разработки обычно представляет собой очередную поставляемую версию программного средства или системы.

Как и при инкрементной стратегии, при реализации эволюционной стратегии зачастую используется прототипирование. В данном случае основной целью прототипирования является обеспечение полного понимания требований

Эволюционная стратегия

Достоинства:

1. возможность **уточнения** и внесения новых требований в процессе разработки;
2. **пригодность** промежуточного продукта для использования;
3. возможность управления **рисками**;
4. обеспечение широкого участия **пользователя** в проекте, начиная с ранних этапов, что минимизирует возможность разногласий между заказчиками и разработчиками и обеспечивает создание продукта высокого качества;
5. реализация преимуществ каскадной и инкрементной стратегий.

Эволюционная стратегия

Недостатки:

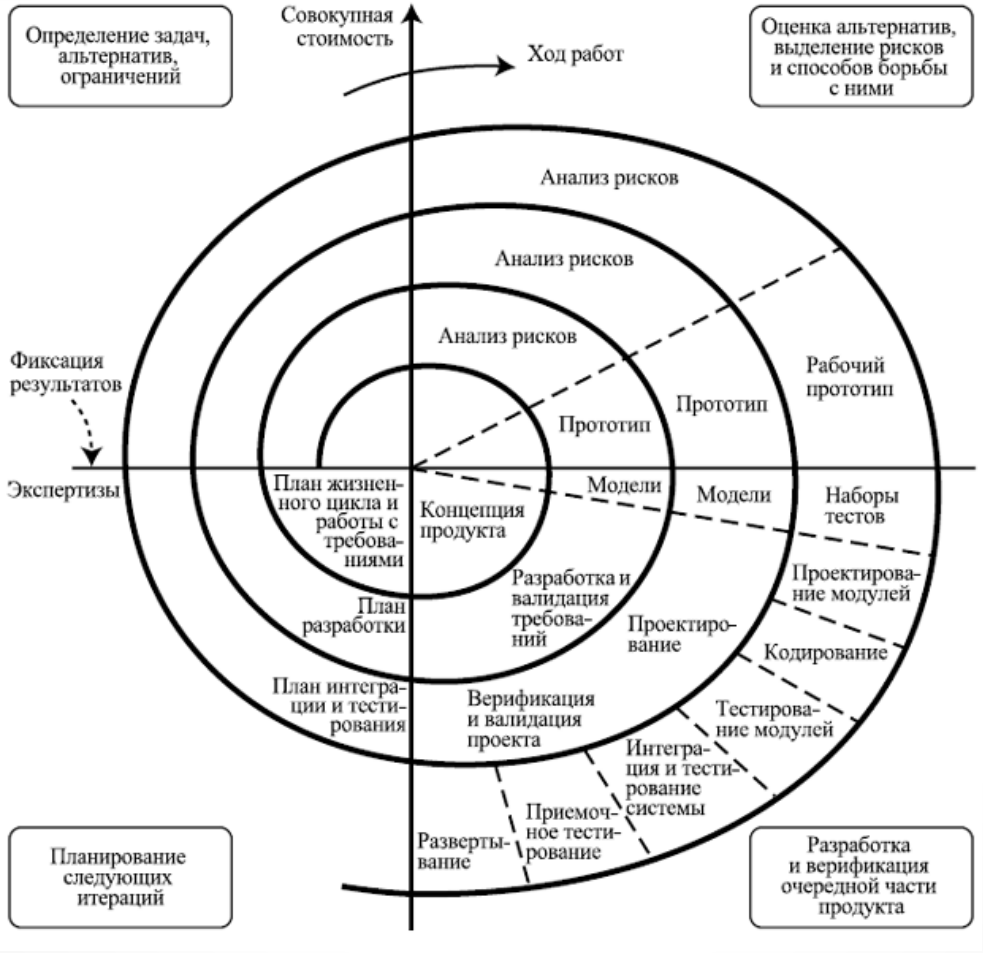
1. **неизвестность** точного количества необходимых итераций и **сложность** определения критериев для продолжения процесса разработки на следующей итерации; это может вызвать задержку реализации конечной версии системы или программного средства;
2. сложность **планирования** и управления проектом;
3. необходимость активного участия **пользователей** в проекте, что реально не всегда осуществимо;
4. необходимость в мощных инструментальных средствах и методах прототипирования;
5. возможность отодвигания решения трудных проблем на последующие циклы, что может привести к несоответствию полученных продуктов требованиям заказчиков.

Эволюционная стратегия

Область применения:

1. при разработке проектов, для которых требования слишком сложны, неизвестны заранее, непостоянны или требуют уточнения;
2. при разработке сложных проектов, в том числе:
    1. больших долгосрочных проектов;
    2. проектов по созданию новых, не имеющих аналогов ПС или систем;
    3. проектов со средней и высокой степенью рисков;
    4. проектов, для которых нужна проверка концепции, демонстрация технической осуществимости или промежуточных продуктов;
3. при разработке проектов, использующих новые технологии.

Спиральная модель



Выбор модели жизненного цикла

Институтом качества программного обеспечения SQI (Software Quality Institute, США) специально для выбора модели ЖЦ предложена схема **классификации** проектов по разработке ПС и систем.

Критерии классификации проектов, предложенные Институтом SQI, объединены в следующие категории:

1. Характеристики **требований к проекту**
2. Характеристики **команды разработчиков**
3. Характеристики **пользователей** (заказчиков)
4. Характеристики типов **проектов и рисков**

Выбор модели жизненного цикла

Критерии категории требований	Каскадная	V-модель	Инкрементная	Эволюционная
Являются ли требования к проекту легко определимыми и реализуемыми?	Да	Да	Нет	Нет
Могут ли требования быть сформулированы в начале ЖЦ?	Да	Да	Да	Нет
Часто ли будут изменяться требования на протяжении ЖЦ?	Нет	Нет	Нет	Да
Нужно ли реализовать основные требования на ранних этапах разработки?	Нет	Нет	Да	Да

Выбор модели жизненного цикла

Критерии категории команды разработчиков проекта	Каскадная	V-модель	Инкрементная	Эволюционная
Являются ли проблемы предметной области проекта новыми для большинства разработчиков?	Нет	Нет	Нет	Да
Изменяются ли роли участников проекта на протяжении ЖЦ?	Нет	Нет	Да	Да
Является ли структура процесса разработки более значимой для разработчиков, чем гибкость?	Да	Да	Да	Нет

Зрелость процессов разработки ПО

Американским университетом Карнеги-Меллон (SoftwareEngineeringInstitute, SEI) разработана модель CMMI (CapabilityMaturityModelIntegration), характеризующая уровни зрелости процесса разработки ПО

- Незрелой называют компанию, где процесс создания ПО и принимаемые решения зависят только от таланта конкретных разработчиков. Результатом является высокий риск превышения бюджета или срыва сроков окончания проекта.
- В зрелой компании работают ясные процедуры управления проектами и построения программных продуктов. По мере необходимости эти процедуры уточняются и развиваются. Оценки длительности и затрат разработки точны, основываются на накопленном опыте. Кроме того, в компании имеются и действуют корпоративные стандарты на процессы взаимодействия с заказчиком, процессы анализа, проектирования, программирования, тестирования и внедрения программных продуктов. Все это создает среду, обеспечивающую качественную разработку программного обеспечения.

Гибкие технологии разработки ПО

Для небольших команд (до 10 участников) альтернативой строго формализованных подходов к разработке ПО являются гибкие (agile) методологии. Гибкие методологии ориентированы на профессионалов, которые мотивированы на создание качественного программного продукта в кратчайшие сроки. Основными положениями гибкого подхода к созданию ПО являются:

- люди и взаимодействие важнее процессов и программных средств;
- работающее ПО важнее исчерпывающей документации;
- взаимодействие с заказчиком важнее согласования условий контакта;
- готовность к изменениям важнее следования первоначальному плану.

Гибкие методологии ориентированы на минимизацию рисков, реализуя короткие итерации длительностью в одну или две недели. Каждая итерация заканчивается выпуском заданной функциональности программного проекта, и реализует этапы работ по планированию, анализу требований, проектированию, кодированию, тестированию и документирование. Отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, но подразумевается что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации, команда выполняет переоценку приоритетов разработки.

Гибкие технологии разработки ПО

Для методологии гибкой разработки декларированы ключевые постулаты, позволяющие командам достигать высокой производительности:

- люди и их взаимодействие;
- доставка работающего программного обеспечения;
- сотрудничество с заказчиком;
- реакция на изменение.

Люди и взаимодействие. Люди - важнейшая составная часть успеха. Отдельные члены команды и хорошие коммуникации важны для высокопроизводительных команд. Для содействия коммуникации гибкие методы предполагают частые обсуждения результатов работы и внесение изменений в решения.

Работающее программное обеспечение важнее всеобъемлющей документации. Все гибкие методологии выделяют необходимость доставки заказчику небольших фрагментов работающего программного обеспечения через заданные интервалы. Программное обеспечение, как правило, должно пройти уровень модульного тестирования, тестирования на уровне системы. При этом объем документации должен быть минимальным. В процессе проектирования команда должна поддерживать в актуальном состоянии короткий документ, содержащий обоснования решения и описание структуры.

Гибкие технологии разработки ПО

Сотрудничество с заказчиком важнее формальных договоренностей по контракту. Чтобы проект успешно завершился, необходимо регулярное и частое общение с заказчиком. Заказчик должен регулярно участвовать в обсуждении принимаемых решений по программному обеспечению, высказывать свои пожелания и замечания. Вовлечение заказчика в процесс разработки программного обеспечения необходимо создания качественного продукта.

Оперативное реагирование на изменения важнее следования плану. Способность реагирования на изменения во многом определяет успех программного проекта. В процессе создания программного продукта очень часто изменяются требования заказчика. Заказчики очень часто точно не знают, чего хотят, до тех пор, пока не увидят работающее программное обеспечение. Гибкие методологии ищут обратную связь от заказчиков в процессе создания программного продукта. Оперативное реагирование на изменения необходимо для создания продукта, который удовлетворит заказчика и обеспечит ценность для бизнеса.

Литература

1. Технология разработки программного обеспечения : учеб. пособие / В. В. Бахтизин, Л. А. Глухова. – Минск : БГУИР, 2010. – 267 с. : ил.
2. Технологии командной разработки программного обеспечения информационных систем http://www.intuit.ru/studies/courses/4806/1054/info
3. Введение в программные системы и их разработку http://www.intuit.ru/studies/courses/3632/874/info