ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

Coding fixing – методика разработки ПО, характеризующаяся отладкой программы и последовательным дополнением ее (в соответствии с результатами отладки).

Главные критерии успешности организации деятельности по разработке ПО:

- Распределение обязанностей
- Единый подход

Следует избегать диспропорции нагрузки

С увеличением количества людей, работающих над проектом, как ни странно, процесс разработки ПО отнюдь не ускоряется. Это связано с лавинообразным нарастанием организационных вопросов (подробнее см [1]). Эта проблема называется **кризис ПО.**

Википедия:

«Кри́зис програ́ммного обеспе́чения» — термин, некогда использовавшийся в информатике для описания последствий быстрого роста вычислительной мощности компьютеров и сложности проблем, которые могут быть решены с их помощью. В сущности, это относится к сложности написания работоспособного, понятного программного обеспечения с использованием верифицированных алгоритмов.

Причины кризиса программного обеспечения были связаны с общей сложностью аппаратного обеспечения и сложностью разработки программного обеспечения. Кризис проявляет себя самым различным образом:

- Стоимость проектов превышает бюджет.
- В проектах превышаются сроки выполнения.
- Программное обеспечение было слишком неэффективным.
- Программное обеспечение имело слишком низкое качество.
- Программное обеспечение зачастую не отвечало необходимым требованиям.
- Проекты были неуправляемыми, и возникали трудности с поддержкой кода.
- Программное обеспечение было непригодным для распространения.

Одна из главных проблем в разработке ПО – это **большая сложность составных частей**. Как правило, ее пытаются преодолеть разными способами, в частности, системным подходом, то есть, учитывая взаимодействие между частями системы, пытаются разобраться в ней.

Программирование занимает 25% от всего процесса разработки ПО.

ХАРАКТЕРИСТИКИ СОВРЕМЕННОГО ПО:

- ✓ Сложность
  - API, библиотеки компонентов, много составных частей
  - Алгоритмы и методы
  - Неспособность одного человека удержать все детали в голове
- ✓ Необходимость реализовать «вчера»
  - o Требования бизнеса в конкурентной среде
- ✓ Низкое повторное использование кода
- ✓ Необходимость интеграции с внешними сервисами
- ✓ Распределенная и неоднородная среда функционирования

Сегодня мы вынуждены выпускать ПО как можно быстрее

Еще одна из трудноразрешимых проблем – необходимость переписывания API под новый язык

ЖИЗНЕННЫЙ ЦИКЛ ПО

Время от идеи до вывода из эксплуатации

Основные этапы

- Разработка требований
  - Правильно сформированные требования – условие того, что все будет правильно сделано
- **Анализ**
- Проектирование
- **Разработка**
- Тестирование
- **Внедрение**
- Эксплуатация
- Вывод из эксплуатации

ДЕЛЕНИЕ ПРОЕКТОВ ПО ОБЪЕМУ

Условное название	Количество человек	Время разработки
Малые	Менее 10	3-6 месяцев
Средние	20-30	1-2 года
Большие	100-300	3-5 лет
Гигантские	1000-2000	7-10 лет

МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА ПО

Структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении всего жизненного цикла.

- Последовательная
  - Определены все требования, один этап разработки
- Инкрементная
  - Определены все требования, несколько этапов разработки
- Эволюционная
  - Определены не все требования, несколько этапов
- Формальных преобразований
  - Определение формальных спецификаций и превращение их в корректные программные интерпретации путем корректного перевода

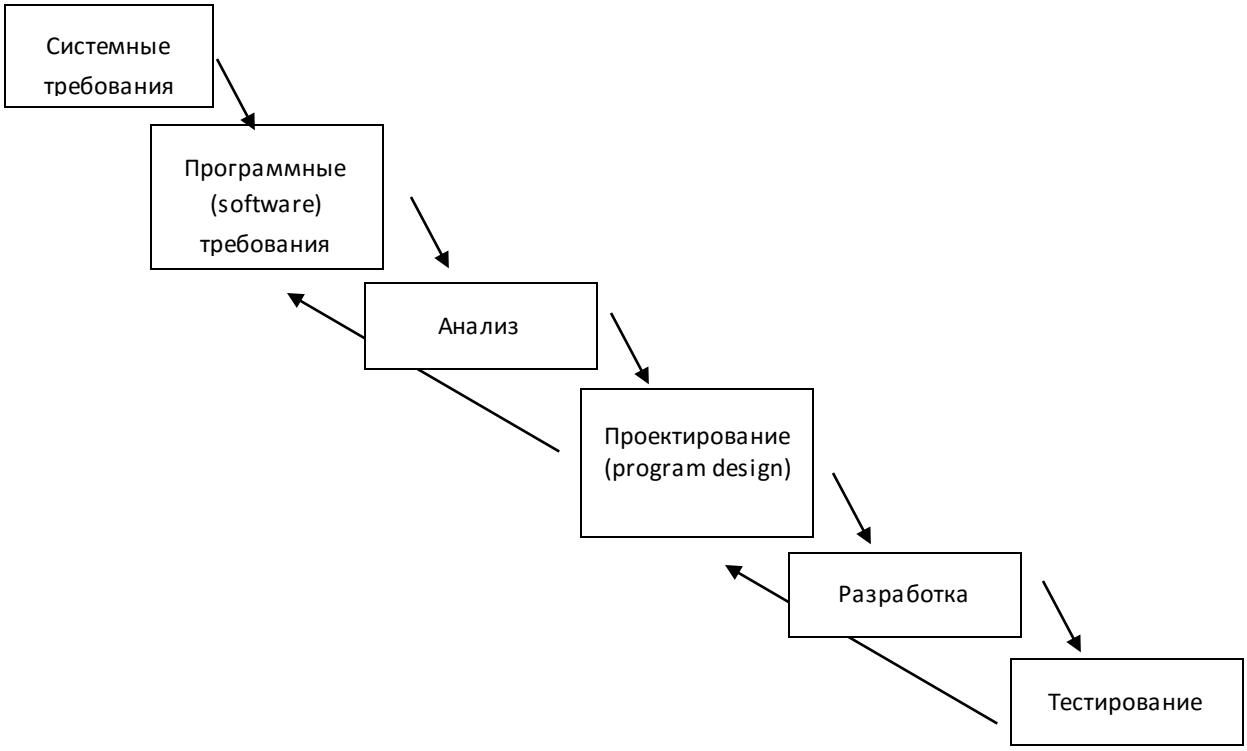
МЕТОДОЛОГИИ РАЗРАБОТКИ

ВОДОПАДНАЯ (КАСКАДНАЯ) МОДЕЛЬ

Разработана в 60-е годы.

Модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

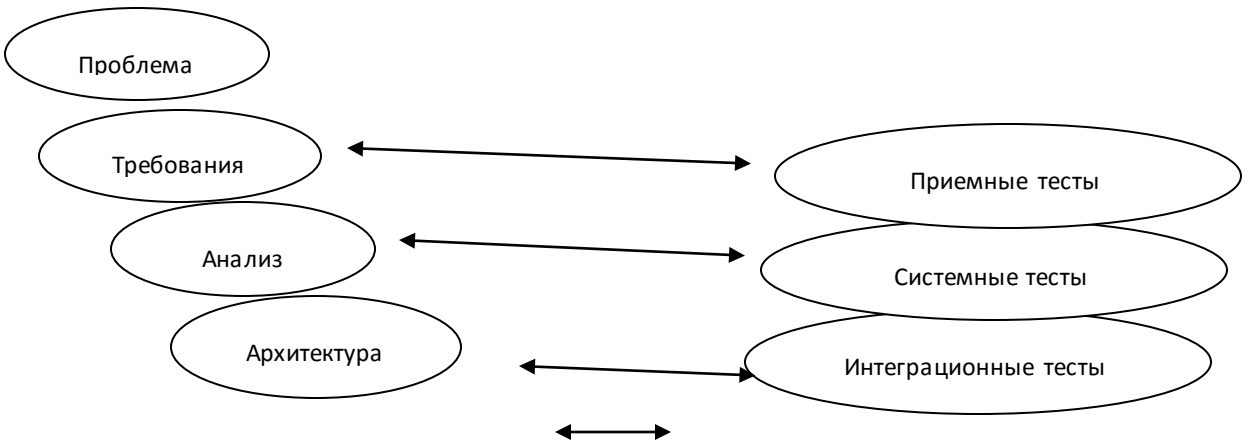
Переход от одной фазы к другой происходит только после полного и успешного завершения предыдущей.

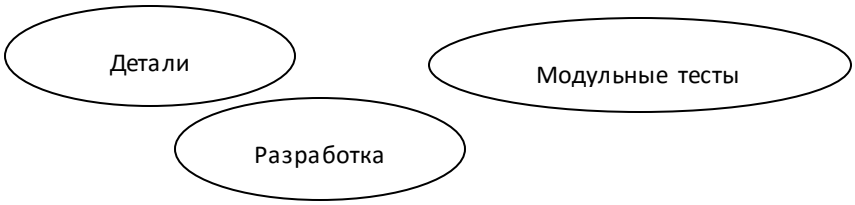


Необходимы при разработке ПО:

- Предварительный анализ
- Документирование
- «do ittwice”
- Сопровождение

ТРАДИЦИОННАЯ V-МОДЕЛЬ





Модульные тесты – тестовый случай

Интеграционные тесты – реализация требований к интерфейсам

Системные тесты – реализация требований к ПО, валидация, верификация

Приемные тесты – бизнес -требования, валидация

Слева на диаграмме – статическое тестирование, справа – динамическое тестирование

Википедия:

Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять. Итерации в проекте производятся по горизонтали, между левой и правой сторонами буквы.

Применительно к разработке информационных систем V-Model — вариация каскадной модели, в которой задачи разработки идут сверху вниз по левой стороне буквы V, а задачи тестирования — вверх по правой стороне буквы V. Внутри V проводятся горизонтальные линии, показывающие, как результаты каждой из фаз разработки влияют на развитие системы тестирования на каждой из фаз тестирования. Модель базируется на том, что приёмо-сдаточные испытания основываются, прежде всего, на требованиях, системное тестирование — на требованиях и архитектуре, комплексное тестирование — на требованиях, архитектуре и интерфейсах, а компонентное тестирование — на требованиях, архитектуре, интерфейсах и алгоритмах.

Цели, которые ставит V-модель:

- **Минимизация рисков:** V-образная модель делает проект более прозрачным и повышает качество контроля проекта путём стандартизации промежуточных целей и описания соответствующих им результатов и ответственных лиц. Это позволяет выявлять отклонения в проекте и риски на ранних стадиях и улучшает качество управления проектов, уменьшая риски.
- **Повышение и гарантии качества:** V-Model — стандартизованная модель разработки, что позволяет добиться от проекта результатов желаемого качества. Промежуточные результаты могут быть проверены на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.
- **Уменьшение общей стоимости проекта:** Ресурсы на разработку, производство, управление и поддержку могут быть заранее просчитаны и проконтролированы. Получаемые результаты также универсальны и легко прогнозируются. Это уменьшает затраты на последующие стадии и проекты.

- **Повышение качества коммуникации между участниками проекта:** Универсальное описание всех элементов и условий облегчает взаимопонимание всех участников проекта. Таким образом, уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком.

МНОГОПРОХОДНАЯ МОДЕЛЬ

Основывается на традиционной V-модели и дополняет ее следующим образом: процесс разработки и сопутствующего детального проектирования и интеграции прототипов разбивается на части и каждая часть впоследствии (результаты каждого прохода) объединяются в один проект.

Важно после разработки все собрать обратно так, чтобы система работала.

RAPID APPLICATION DEVELOPMENT (RAD)

- ✓ Конструкторы
  - o Пользователи принимают участие в конструировании
- ✓ Взаимодействие с пользователем
- ✓ Технологии

Википедия:

Концепция создания средств разработки программных продуктов, уделяющая особое внимание быстроте и удобству программирования, созданию технологического процесса, позволяющего программисту максимально быстро создавать компьютерные программы. Практическое определение: RAD — это жизненный цикл процесса проектирования, созданный для достижения более высокой скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию. С конца XX века RAD получила широкое распространение и одобрение. Концепцию RAD также часто связывают с концепцией визуального программирования.

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

RAD методология применима в случаях:

1. *Необходимо выполнение проекта в сжатые сроки.* Быстрое выполнение проекта позволяет создать систему, отвечающую требованиям сегодняшнего дня. Если система проектируется долго, то весьма высока вероятность, что за это время существенно изменятся фундаментальные положения, регламентирующие деятельность организации, то есть, система морально устареет ещё до завершения её проектирования.

2. *Нечетко определены требования к ПО.* В большинстве случаев заказчик весьма приблизительно представляет себе работу будущего программного продукта и не может четко сформулировать все требования к ПО. Требования могут быть вообще не определены к началу проекта либо могут изменяться по ходу его выполнения.

3. *Проект выполняется в условиях ограниченности бюджета.* Разработка ведётся небольшими RAD-группами в короткие сроки, что обеспечивает минимум трудозатрат и позволяет вписаться в бюджетные ограничения.

4. *Интерфейс пользователя (GUI) есть главный фактор.* Нет смысла заставлять пользователя рисовать картинки. RAD-технология дает возможность продемонстрировать интерфейс в прототипе, причём достаточно скоро после начала проекта.

5. *Возможно разбиение проекта на функциональные компоненты.* Если предполагаемая система велика, необходимо, чтобы её можно было разбить на мелкие части, каждая из которых обладает четкой функциональностью. Они могут выпускаться последовательно или параллельно (в последнем случае привлекается несколько RAD-групп).

6. *Низкая вычислительная сложность ПО.*

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

- Инструментарий должен быть нацелен на минимизацию времени разработки.
- Создание *прототипа* для уточнения требований заказчика.
- Цикличность разработки: каждая новая версия продукта основывается на оценке результата работы предыдущей версии заказчиком.
- Минимизация времени разработки версии, за счёт переноса уже готовых модулей и добавления функциональности в новую версию.
- Команда разработчиков должна тесно сотрудничать, каждый участник должен быть готов выполнять несколько обязанностей.
- Управление проектом должно минимизировать длительность цикла разработки.

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

Технология *быстрой разработки приложений* (RAD) позволяет обеспечить:

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

RAD использует CASE – системы, которые

- позволяют строить модели и превращать их в заготовки кода, обрабатывать требования
- используются в крупных бизнес – системах SAP, Oracle, E-business suite
- Любые средства моделирования и поддержка процесса

RATIONAL UNIFIED PROCESS (RUP)

UP методологии - 90-е годы

RUP – 1998 год

Фазы RUP

- Начало
  - Цели
    - Определить границы проекта
    - Описать основные сценарии использования системы
    - Предложить возможные технические решения
    - Посчитать стоимость и разработать график работ
    - Оценить риски, подготовить окружение
  - Веха «Lifecycle Object"
    - Согласие сторон в оценке сроков, первоначальной стоимости, требованиям, приоритете, технологиях
    - Оценены риски и выбраны стратегии смягчения последствий
- Проектирование
  - Цели
    - Финализировать базовую архитектуру системы
    - Разработать прототипы на основе архитектуры
    - Убедиться в том, что архитектура, планы и сроки стабильны, риски разработаны и учтены
    - Продемонстрировать, что в архитектуре можно будет реализовать требования с разными сроками и стоимостью
- Построение
  - Цели
    - Экономически эффективно, с надлежащим качеством, так быстро, как возможно, разработать продукт
    - Итеративно и инкрементивно провести анализ, проектирование, разработку и тестирование продукта, создать необходимые выпуски продукта
    - Подготовить продукт, места установки и пользователей к использованию
  - Веха «Initial Operational Capability"
    - Выпуск достаточно стабилен для передачи пользователям
    - Все стороны готовы к передаче продукта пользователям
    - Соотношение запланированных и затраченных расходов все еще приемлемо
- Внедрение
  - Цели
    - Провести бета тестирование, сравнить работоспособность старой и новых версий
    - Перенести продуктивную БД, обучить пользователей и поддерживающий персонал
    - Запустить маркетинг и продажи
    - Отладить процессы устранения сбоев, дефектов, проблем с производительностью
    - Убедиться в самодостаточности пользователей
    - Провести открытый анализ совместно со всеми заинтересованными сторонами соответствия концепции разработанному продукту
  - Веха «Product Release"
    - Пользователи удовлетворены

- Финальное соотношение запланированных и затраченных расходов приемлемо

Дух RUP

- ✓ Атаковать риски как можно раньше
- ✓ Обеспечить выполнение требований именно заказчиков
- ✓ Концентрироваться на исполняемом коде
- ✓ Готовиться к изменениям с самого начала
- ✓ Составлять систему из компонент
- ✓ Разработать архитектуру как можно раньше
- ✓ Сделать качество основной идеей

Agile manifesto (2001)

- Люди и взаимодействие важнее процессов и инструментов
- Работающий продукт важнее исчерпывающей документации
- Сотрудничество с заказчиком важнее согласования условий контракта
- Готовность к изменениям важнее следования первоначальному плану

SCRUM

Работает только на маленьких проектах.

Википедия:

Методология управления проектами, применяющаяся при необходимости гибкой разработки. Методология делает акцент на качественном контроле процесса разработки.

Scrum — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость.

Спринт — итерация в скраме, в ходе которой создаётся функциональный рост программного обеспечения. Жёстко фиксирован по времени. Длительность одного спринта от 2 до 4 недель. В отдельных случаях, к примеру согласно скрам-стандарту компании Nokia, длительность спринта должна быть не более 6 недель. Тем не менее, считается, что чем короче спринт, тем более гибким является процесс разработки, релизы выходят чаще, быстрее поступают отзывы от потребителя, меньше времени тратится на работу в неправильном направлении. С другой стороны, при более длительных спринтах команда имеет больше времени на решение возникших в процессе проблем, а владелец проекта уменьшает издержки на совещания, демонстрации продукта и т. п. Разные команды подбирают длину спринта согласно специфике своей работы, составу команд и требований, часто методом проб и ошибок. Для оценки объёма работ в спринте можно использовать предварительную оценку, измеряемую в очках истории. Предварительная оценка фиксируется в бэклоге проекта.

Основные роли (Core roles) в методологии скрам («Свиньи»)

«Свиньи» полностью включены в проект и в скрам-процесс (**Scrum Team**).

- **Скрам-мастер (Scrum Master)** — проводит совещания (Scrum meetings) следит за соблюдением всех принципов скрама, разрешает противоречия и защищает команду от отвлекающих факторов. Данная роль не предполагает ничего иного, кроме корректного ведения скрам-процесса. Руководитель проекта скорее относится к **владельцу проекта** и не должен фигурировать в качестве скрам-мастера.

- **Владелец продукта (Product Owner)** — представляет интересы конечных пользователей и других заинтересованных в продукте сторон.

- **Команда Разработки (Development Team)** — кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: тестировщиков, архитекторов, аналитиков, программистов и т. д. Размер команды в идеале составляет от 3 до 9 человек. Команда является единственным полностью вовлечённым участником разработки и отвечает за результат как единое целое. Никто, кроме команды не может вмешиваться в процесс разработки на протяжении спринта.

Дополнительные роли (Ancillary roles) в методологии скрам («Куры»)

- **Пользователи (*Users*)**

- **Клиенты, Продавцы (*Stakeholders*)** — лица, которые инициируют проект и для кого проект будет приносить выгоду. Они вовлечены в скрам только во время обзорного совещания по спринту (Sprint Review).

- **Управляющие (*Managers*)** — люди, которые управляют персоналом.

- **Эксперты-консультанты (*Consulting Experts*)**

РАЗРАБОТКА ТРЕБОВАНИЙ

Требования

- ✓ Условия или возможности, которыми должна обладать система
- ✓ Подробное описание того, что должно быть реализовано
  - Не описание того, как должно быть реализовано
- ✓ Описываются с помощью
  - Модели требований (шаблон SRS из RUP)
  - Модели прецедентов

Свойство требования

- Корректность
- Однозначность
- Полнота
- Непротиворечивость
- Приоретизация
- Проверяемость

- Модифицируемость
- Отслеживаемость

Классификация требований FURPS+

- Functional
- Usability
- Reliability
- Performance
- Supportability
- Дополнительные (+)
    - Ограничения проектирования \ архитектуры
    - Требования к реализации
    - Физические требования

Идентификация требования:

<id><система>должна/shall<требование>

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ (FUNCTIONAL)

Определяют:

- ✓ Feature sets
- ✓ Capabilities – возможности ПО
- ✓ Security – требования к безопасности

Пример:

FR0 Система должна обеспечивать ввод, модификацию и удаление данных о клиенте

SEC0 Система должна обеспечивать двухфакторную аутентификацию пользователей с помощью имени пользователя и пароля и подтверждение по СМС

USABILITY

- Human factors – учет особенностей пользователя
    - Система не должна работать слишком быстро или слишком медленно
- Aesthetic – эстетические требования
- Consistency in the UI – Согласованность пользовательского интерфейса
- Online and context-sensitive help – требования к справочной подсистеме
- Wizards and agents – мастера и ПО, повышающие продуктивность и простоту работы пользователя
- User documentation – требования к документации пользователя
- Training materials – требования к учебным материалам

RELIABILITY (НАДЕЖНОСТЬ)

- Frequency and severity of failure – частота и обработка отказов
- Recoverability – способность системы восстанавливать продуктивное функционирование

- Predictability – предсказуемость поведения
- Accuracy – точность

MTBF (Mid time between failures – среднее время между отказами)

PERFORMANCE (ПРОИЗВОДИТЕЛЬНОСТЬ)

- Speed – скорость решения задач
- Efficiency – эффективность
- Availability – готовность системы к решению задач
- Throughput – пропускная способность
- Response time – время отклика
- Recovery time – время восстановления
- Resource usage – использование ресурсов

SUPPORTABILITY

- Testability – проверяемость результатов
- Extensibility – расширяемость
- Adoptibility – адоптируемость под конкретные задачи
- Maintainability – поддерживаемость
- Compatibility – совместимость

АТРИБУТЫ ТРЕБОВАНИЙ

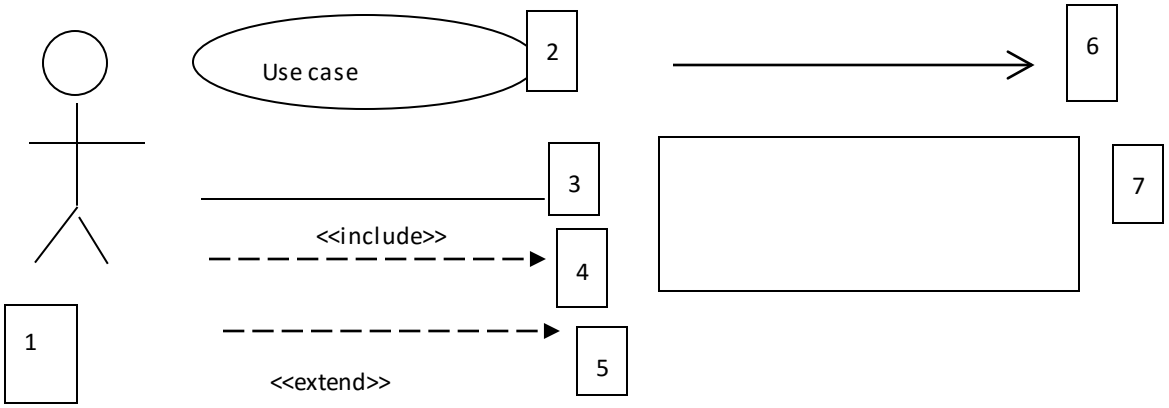
- ✓ Приоритет – MoSCoW
  - MUST have (Minimum Usable SubseT) – фундаментальные для системы требования
  - Should have – важные
  - Could have – потенциально возможные, улучшающие, к примеру, пользовательские отношения
  - Want to have (Would like to have) – могут быть реализованы в следующих версиях системы
- ✓ Дополнительно используются цифровые приоритеты
- ✓ Статус
  - Предложенные
  - Одобренные
  - Отклоненные
  - Включенные
- ✓ Трудоемкость
  - Человеко-часы
  - Функциональные точки
  - Попугаи
- ✓ Риски
- ✓ Стабильность
  - Высокая
  - Средняя
  - Низкая
- ✓ Целевая версия

Поиск требований:

- Пользователи
- Руководители
- Специалисты по установке
- Другие системы, аппаратные ограничения
- Технические и правовые ограничения
- Коммерческие цели

UML : Use-case model

- ✓ Actor – актер (1)
- ✓ Usecase – прецедент использования (2)
- ✓ Associacion – ассоциация, использование (3)
- ✓ Include – включение (4)
- ✓ Extend – точка расширения функционала (5)
- ✓ Generalization – обобщение (6)
- ✓ System boundary – границы системы (7)



Прецедент:

PieSelling
ID:2
Краткое описание: Бабушка продает пирожки
Главные актеры: Бабушка-продавец, клиент
Второстепенные актеры: нет
Предусловия: …
Основной поток: …

РИСКИ

- Риск – потенциаьно опасный (для проекта) фактор
- Менеджмент риска. Принципы и руководство
    - Риск – сочетание вероятности события и его (негативных) последствий

Типы рисков

- ✓ Прямые и непрямые
    - Можно управлять, контролировать риск или нет
- ✓ Ресурсные
- ✓ Бизнес-риски
- ✓ Технические риски
- ✓ Политические риски
- ✓ Форс-мажор

Управление рисками

- ➢ Оценка (assessment) риска
    - Идентификация риска
        - Известные риски
        - Неизвестные риски
        - Непознаваемые риски
    - Анализ риска
        - Различные модели и методы анализа
            - Стоимости
            - Сетевой
            - Качественных факторов
        - Вероятность и масштаб
            - Могут быть заданы нечетко
    - Приоритизация риска
        - Экспозиция риска RE=Prob(UO)*Loss(UO)
        - Другие факторы из анализа
        - Создается документ «TOP-10”
- ➢ Контроль и управление риском
    - Планирование управления (реакции) на риски

УПРАВЛЕНИЕ ИЗМЕНЕНИЯМИ ПРОГРАММНЫХ ПРОДУКТОВ

Изменение – это контролируемое, журналируемое обновление системы (изменения необходимо уметь контролировать)

- ✓ Атрибуты изменений
    - Идентификатор
    - Дата
    - Ответственный

- Описание
- Журнал изменения

✓ Связаны с запросом на изменение:
- Требование новой функциональности
- Нефункциональные требования
- Исправление дефекта

СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ

- Предназначены для управления изменениями в программном коде
- Поддерживают групповую работу
- Основные типы:
  - На основе файловой системы (с экспортом клиентам – очень старые)
  - Централизованные (один центральный репозиторий на сервере, который указывается явным образом)
  - Распределенные (репозиторий на каждом клиенте с центральным репозиторием на сервере, который не обозначается явным образом, что именно он является центральным)
- Клиенты встроены в большинство средств разработки
- RSC, ClearCase, CVS, Subversion, MS Visual Source Safe, mercurial, git, Bazaar, …

Одновременная модификация файлов:

✓ Lock – modify – unlock
- В основном используется в СКВ на файловых системах
- Замедляет работу команды

✓ Copy – modify – merge
- Своя рабочая копия у каждого разработчика
- Трудности слияния рабочих копий

SVN

- Централизованная СКВ, репозиторий хранится на сервере
- Доступ к репозиторию осуществляется при помощи:
  - Sunserve
  - Apache + модуль mod_div_svn
- Каждый входящий коммит повышает версию репозитория на единицу
- Trunk – основная ветвь разработки
- Branches – хранение модификации продукта
  - Releases – значимые версии продукта (релизы)
  - Features – выполнение работ над версиями с существенными изменениями без влияния на trunk
  - Vendor – версии с модификациями сторонних библиотек
- Tag – функционально целостное изменение
  - Исправления дефектов ПО
  - Минорные версии

Основной цикл разработки:

`svn checkout` – первоначальное создание рабочей копии

`svn update` – обновление рабочей копии (загрузка изменений с сервера)

`svn add / delete / copy / make / mkdir // status / diff // revert` – изменение необходимых файлов

`svn update` – обновление рабочей копии (загрузка новых изменений с сервера непосредственно перед коммитом)

Возможно, перед коммитом также потребуется разрешить конфликты содержимого и структуры файлов:

- ✓ e (edit) – изменить файл в редакторе
- ✓ df (diff-full) – показать список несовпадений между весиями
- ✓ r (resolved) – подтвердить, что конфликт разрешен в текущей (может быть редактированной) версии файла
- ✓ dc (display conflict) – показать все конфликты
- ✓ …
- ✓ p (postpone) – отложить разрешение конфликта (в рабочей директории остаются несколько новых файлов)
    - o .mine – до апдейта
    - o .rOLDEREV – до правок (старая ревизия)
    - o rNEWREV – версия из репозитория (новая ревизия)

Что касается конфликтов структуры:

- ✓ Происходят, когда в репозитории перемещаются файлы, а в рабочей копии изменяются

`svn commit` – фиксация сделанных изменений

Слияние изменений из веток происходят с помощью команды merge.
`svn merge ^/branches/FR217-newmovies r10:HEAD` – подгружает изменения из веток в рабочую копию, учитывая изменения структуры

GIT

Разделяет такие понятия, как

- ✓ Исходный (forked) репозиторий – репозиторий, от которого осуществлено ответвление (fork) и создан наш собственный проект
- ✓ Удаленный репозиторий – репозиторий, который хранится на сервере и неявно считается главным
- ✓ Локальный репозиторий – расположенный на компьютере разработчика и с котором работает программист в промежутках между коммитами на сервер
- ✓ Рабочая копия – директория, содержащая файлы, которые можно непосредственно удалять или редактировать, но которые отделены от локального репозитория
- ✓ Stage Area – место, содержащее в себе информацию о том, какие изменения рабочей копии должны быть зафиксированы в локальном репозитории при следующем коммите

Считается хорошим стилем использование регламентированных веток:

- Master – последняя рабочая версия продукта, которую в любой момент можно запустить
- Develop – предназначена для разработки системы
- Features – разработка отдельных частей системы
- Release – сюда идут только те изменения, которые будут фиксироваться в master
- Hotfix – ветка для устранения критических дефектов

Git flow – плагин для Git – версионирование в терминах версий, а не операций – упрощает работу с СКВ

Git flow [init / [release hotfix feature [start finish publish pull] NAME]

SVN требует меньшего набора команд для совершения тех же действий, что и GIT