Технологии разработки программного обеспечения

Процессы разработки ПО

Процессы разработки ПО (software processes)

Программная инженерия (Software engineering)

- Программная инженерия это систематизированный подход к профессиональной разработке, внедрению, сопровождению и изъятию из использования ПО.
- Такой систематизированный подход должен помогать повышать качество и производительность разработки ПО (К&П).

- Качество и производительность разработки ПО зависит от:
  - ***квалификации (умения) людей,*** участвующих в разработке ПО;
  - ***качества процессов*** организации работы специалистов, которые используют для выполнения различных задач проекта;
  - возможностей используемых ***программных средств разработки*** (инструментов).

Типы программных систем

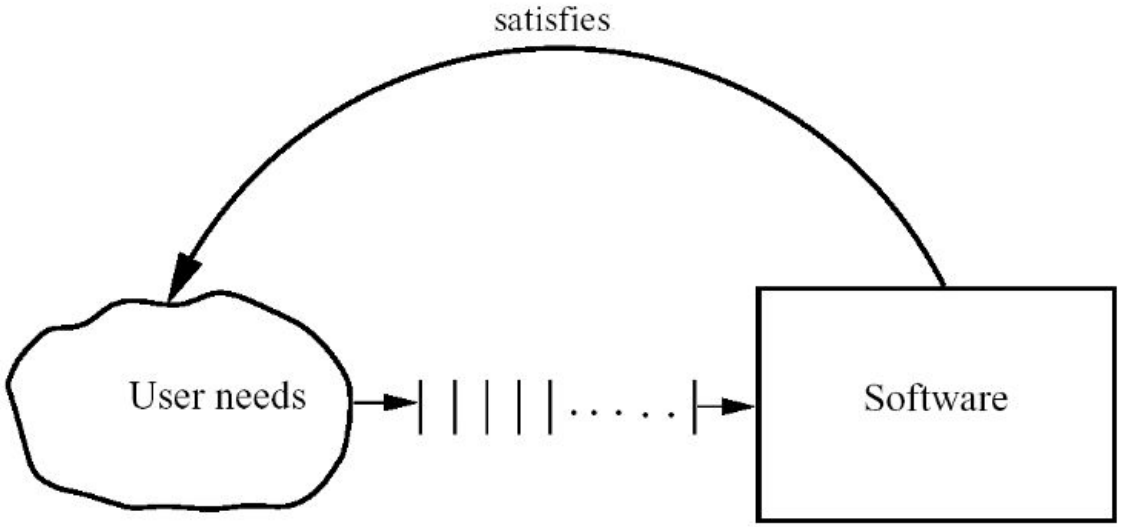
1. системы работающие только с людьми
   – например: информационные системы организаций – используются людьми для ввода, хранения поиска и отображения информации;
2. системы работающие с людьми и внешними техническими устройствами:
   – например: информационные системы соревнований.
3. системы работающие с внешними техническими устройствами
   – встроенные системы реального времени – встраиваются в технические устройства для выполнения управления ими.

Процессы – основа разработки ПО

- В ТРПО основное внимание уделяется на процессы, которые называются систематизированном подходом к разработке ПО.
- Основной задачей ***процессов*** является оказание помощи специалистам достигать высокого качества и производительности путем указания того, какие задачи и как требуется решать.
- Внимание к процессу создания программного продукта и отличает ТРПО от большинства других компьютерных дисциплин.
- Процессы (совместно с инструментами и технологиями для их выполнения) составляют основу ТРПО и поэтому в данной дисциплине

Процесс и проект (process and project)

- Процесс это последовательность шагов, выполняемых для достижения требуемой цели.
- При разработке ПО промышленного уровня целью является создание ПО, удовлетворяющего потребностям пользователей.



- Для программного проекта ключевую роль играет процесс разработки – в результате выполнение данного процесса достигается цель предоставления заказчику требуемого ПО.
- Однако не достаточно просто разработать требуемое ПО, нужно выполнить проект с ***наименьшими затратами***, за ***наименьшее время*** и с ***требуемым качеством***.
- В связи с этими дополнительными целями роль процессов возрастает.
- Существует много процессов разработки, позволяющих создать требуемое ПО, но ***для достижения высокого качества и производительности требуется некоторый «оптимальный» процесс***.

Спецификация процесса

- Нужно отличать спецификацию (детальное описание) процесса от самого процесса.
- ***Процесс*** является динамической сущностью, включающей выполненные действия.
- ***Спецификация процесса*** является описанием процесса, которому предположительно можно следовать в некотором проекте для достижения цели, для которой данный процесс был спроектирован.
- В проекте спецификация процесса может использоваться, в качестве процесса, который планирует использовать проект.
- Реальный процесс это то, что в действительно выполняется в проекте.
- Он может отличаться планируемого процесса.

Модель процесса

- ***Модель процесса*** описывает (специфицирует) обобщенный процесс*, который является «оптимальным» для* ***некоторого класса проектов***.
- Т.е. в ситуациях, для которых данная модель является применимой, использование данной модели процесса приведет к достижению цели разработки ПО с высокими показателями К&П.
- Модель процесса фактически является объединением лучших практических решений в некоторый «рецепт» успешного выполнения проекта.
- Процесс часто специфицируется (детально описывается) в виде последовательностей высокоуровневых этапов.
- В свою очередь каждый из этапов состоит из более мелких шагов и иногда называется подпроцессом.

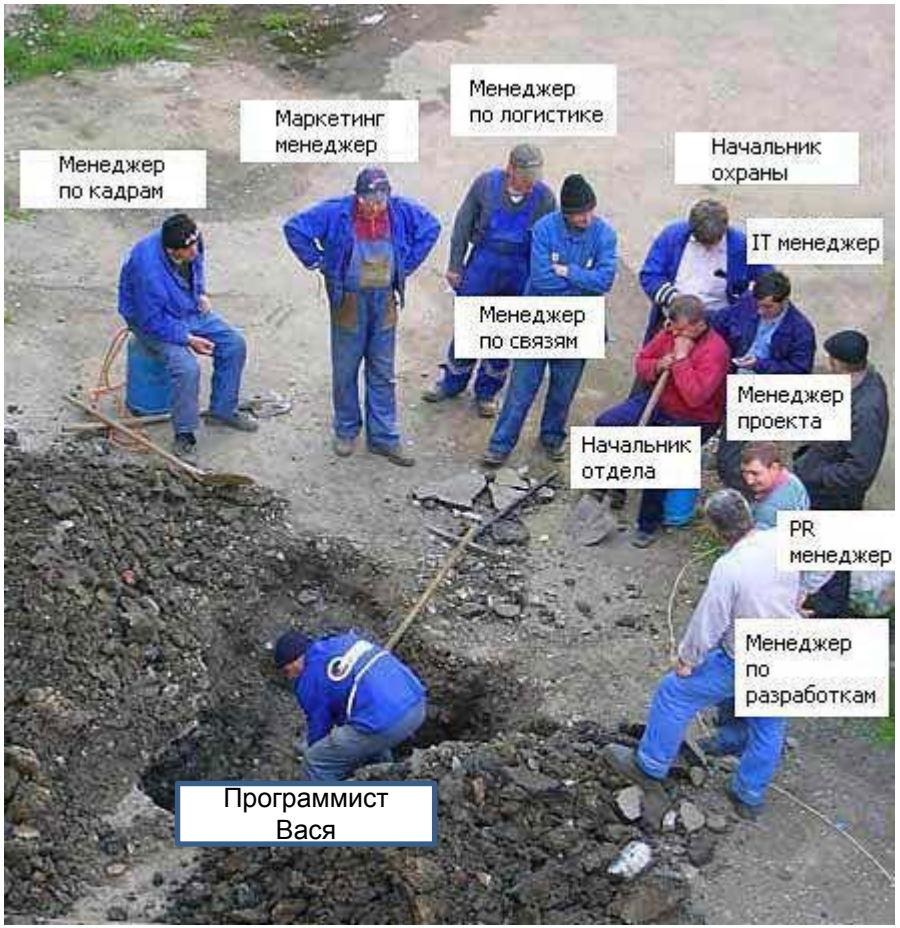
Процессы разработки ПО

- Т.к. при разработке ПО требуется достижение разных целей, то требуются разные процессы.
- Многие из них не касаются ТРПО, но влияют на разработку ПО.
- Все процессы можно разделить на две большие группы:
  - Процессы не связанные непосредственно с разработкой ПО
  - Процессы непосредственно связанные с разработкой ПО.

Процессы не связанные непосредственно с разработкой ПО

- ***бизнес-процессы*** –
  - поиска заказчиков,
  - ведение переговоров,
  - убеждение в необходимости заключения договора,
  - оформление договора,
  - договор о цене,
  - оформление оплаты;
- ***социальные процессы*** –
  - организация команд,
  - создание творческой атмосферы,
  - праздничные вечера,
  - коллективные поездки;
- ***процесс обучения*** – лекции, мастер-классы, участие в конференциях.
- ***реклама***
- ***научная работа***
- и пр.

Команда разработчиков ПО



Программные процессы (software process)

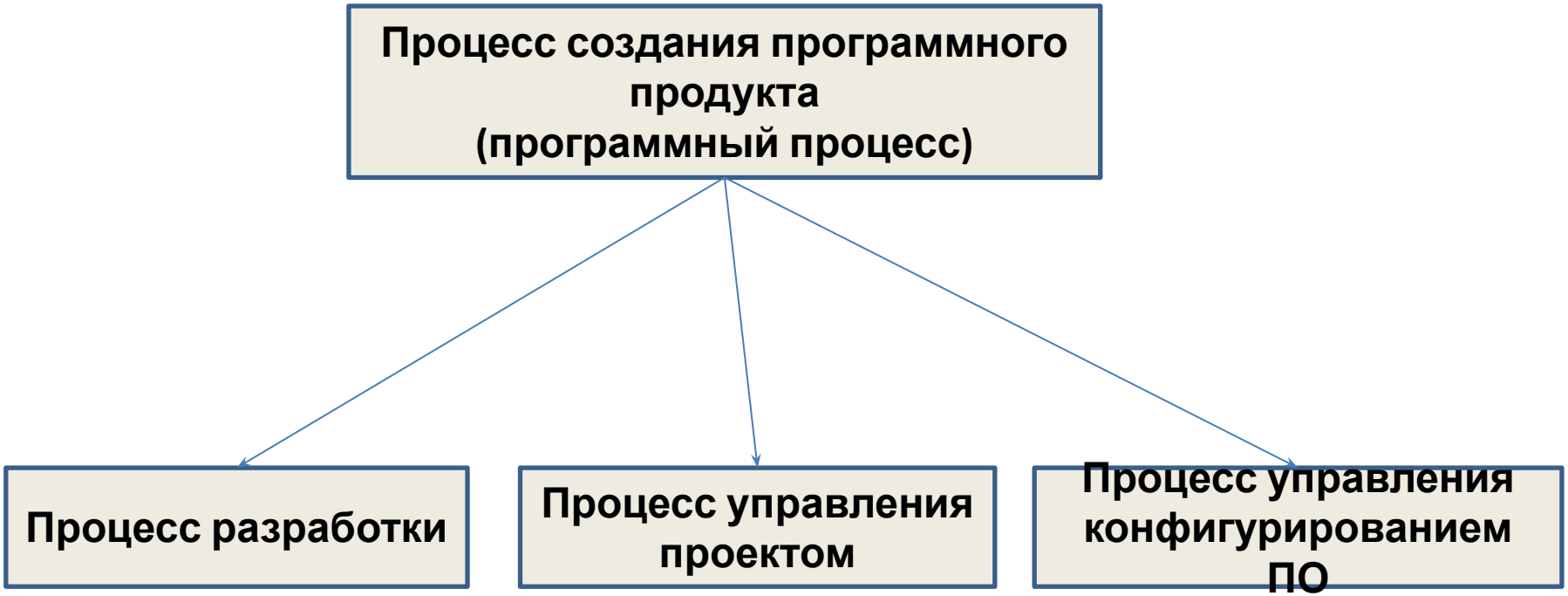
- *Процессы, который непосредственно имеет дело с техническими и управленческими задачами разработки ПО в общем называются* ***программным процессом*** *(**software process**)*.
- Разработка ПО разделена по проектам – в каждом проекте создается ПО для конкретного заказчика.

Программный процесс

- Программный проект (software project) должен
  1. разрабатывать программное обеспечение;
  2. выполнять правильное управление данным проектом.
- В связи с этим в программный процесс состоит из двух основных подпроцессов:
  1. процесс разработки – определяет все требуемые виды инженерной деятельности по созданию ПО;
  2. процесс управления проектом – определяет то, как планируются и управляются эти виды деятельности.
- Процессы эффективной разработки ПО и управления проектом являются основными факторами достижения цели предоставления желаемого ПО, удовлетворяющего требованиям заказчика, при обеспечении высокой производительности и качества.

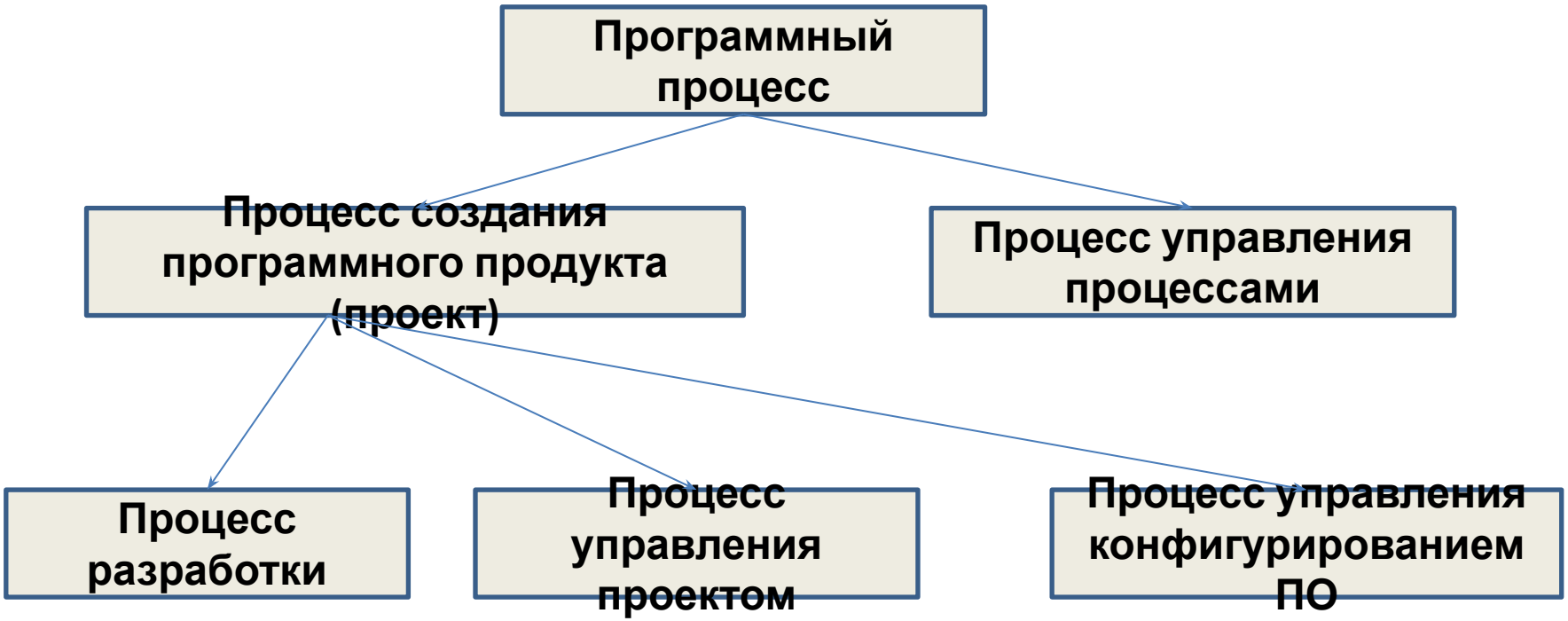
Программный процесс

- Подпроцессы программного процесса



Программный процесс

- Подпроцессы программного процесса



Подпроцесс управления конфигурированием ПО

- В проекте разрабатываются много разных компонент (например, конечный исходный код может состоять из большого числа исходных файлов).
- Эти компоненты развиваются по мере выполнения проекта, создается большое число их версий.
- Для управления развитием и изменениями часто используется подпроцесс управления конфигурированием ПО.
- Данный подпроцесс в основном связан с таким управлением изменениями, чтобы при этом не нарушалась целостность программных продуктов.

Процесс управления процессами разработки ПО

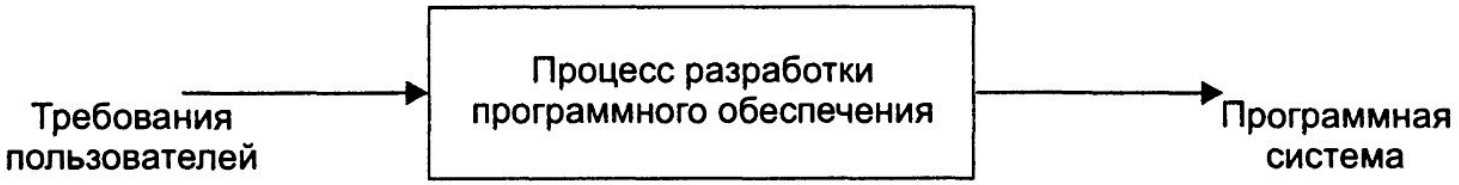
- Сами программные процессы изменяются, развиваются, чтобы приспособиться к улучшению понимания разработки ПО и доступности новых технологий и инструментов
- Основная цель процесс управления процессами разработки ПО является улучшение программного процесса.
- Под улучшением программного процесса понимается их совершенствование для разработки качественных продуктов при низких затратах.

Процесс разработки ПО

Процесс разработки ПО

- Процесса разработки ПО является базовым процессом и его целью является создание высококачественного программного продукта.
- Он включает работы, непосредственно связанные с созданием ПО (такие, как проектирование, кодирование и тестирование).
- Процесс управления принимает решения на основе результатов работы процесса разработки.
- **Процесса разработки ПО определяет то, какие задачи должны решаться в проекте и в каком порядке они должны выполняться.**
- Он ограничивает свободу выполнения проекта, таким образом, что «кратчайшим» путем (или даже наиболее эффективным) перейти от требований потребителя к ПО, которое им удовлетворяет
- Данный процесс направляет проект и существенно влияет на полученные результаты.

- **Процесс разработки программного обеспечения**



Жизненный цикл ПО

- **Жизненный цикл** ПО это период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.
- Этот цикл включает процесс построения и развития ПО.
  1. Выявление и спецификация требований.
  2. Анализ
  3. Проектирование
  4. Кодирование
  5. Тестирование
  6. Передача заказчику
  7. Сопровождение
  8. Развитие

Модели процессов разработки ПО

- Модель процесса (МП) описывает обобщенный процесс, и обычно включает:
  - набор этапов, на которые должен быть разделен процесс;
  - порядок, в котором эти процессы должны выполняться;
  - разные ограничения и условия на выполнение этих этапов.
- Основное предположение для модели процесса заключается в том, что в тех ситуациях, когда данная модель является применимой, ее использование в качестве процесса проекта позволит:
  1. уменьшить расходы,
  2. повысить качество,
  3. уменьшить время разработки,
  4. а также принести другие выгоды.
- Другими словами, модель процесса предоставляет общие методические указания для разработки подходящего процесса выполнения проекта.

- Основными моделями процесса разработки ПО являются следующие:

1. Модель водопада
2. Прототипирование
3. Итеративная разработка
4. Rational Unified Process
5. Модель временных ящиков
6. Гибкая разработка.

Модель процесса

- ***Модель процесса*** описывает (специфицирует) общий процесс*, который является «оптимальным» для* ***некоторого класса проектов***.
- Т.е. в ситуациях, для которых данная модель является применимой, использование данной модели процесса приведет к достижению цели разработки ПО с высокими показателями К&П.
- Модель процесса фактически является объединением лучших практических решений в некоторый «рецепт» успешного выполнения проекта.
- Процесс часто специфицируется (детально описывается) в виде последовательностей высокоуровневых этапов.
- В свою очередь каждый из этапов состоит из более мелких шагов и иногда называется подпроцессом.

1. Модель водопада

- Самой простой моделью процесса является модель водопада, в соответствии с которой все этапы организованы в линейном порядке.
- Имеется много вариантов данной модели, включающих разные виды работ и разный поток управления между ними.
- В данной модели проект начинается с ***анализа осуществимости***.
- При успешной демонстрации осуществимости проекта, начинается ***анализ требований и планирование проекта***.
- После завершения анализа требований начинается ***проектирование***, а после него начинается ***составление (кодирование) программы***.
- После успешного завершения программирования, созданный код объединяется и выполняется тестирование.
- После успешного завершения тестирования, система устанавливается у заказчика.
- После этого выполняется постоянное использование созданного ПО и его поддержка.

Модель водопада

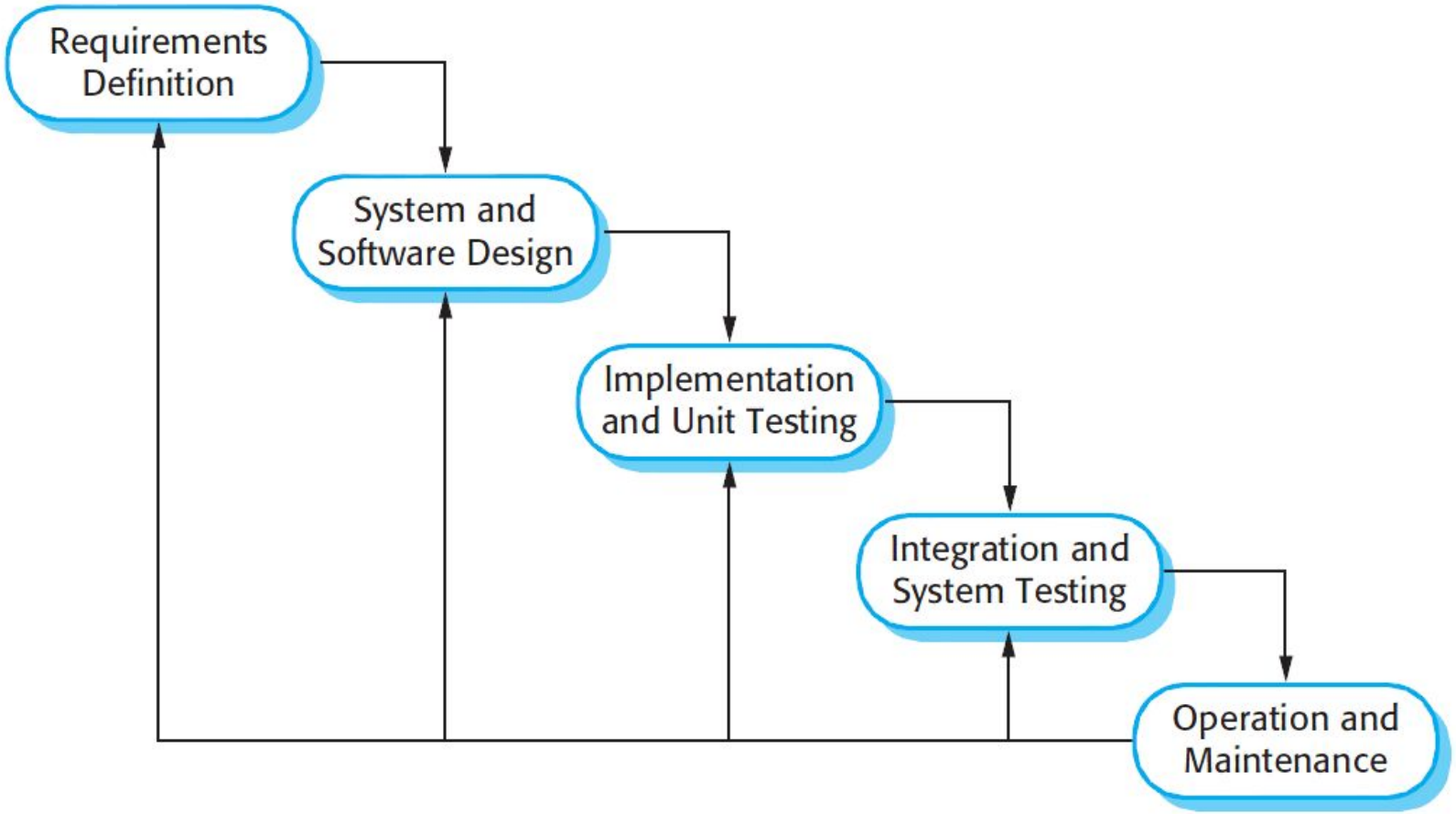
- На данной схеме этап анализа требований назван «анализом и планированием».
- Планирование является критически важной работой для разработки ПО.
- Хороший план основывается на требованиях системы и должен выполняться до того, как начнутся другие этапы проекта.
- Однако на практике, детальные требования не обязательны для планирования, поэтому планирование обычно выполняется одновременно с анализом требований и план подготавливается до того, как начнутся другие этапы проекта.
- Разработанный план является дополнительными исходными данными для всех последующих этапов.

- Линейный порядок работ имеет некоторые важные последствия.
- Для явного распознавания завершения очередного этапа и начала следующего, требуются использовать некоторый способ подтверждения в конце каждого этапа.
- Это обычно выполняется с помощью некоторой проверки (verification) и утверждения (validation), которые будут гарантировать, что:
  - результат выполнения этапа согласуется с его входными данными (что являлось результатом работы (выходом) предыдущего этапа);
  - результат выполнения данного этапа согласуется с общими требованиями заказчика к системе.
- Следствием потребности в подтверждении является то, что каждый этап должен иметь конкретный результат, который м.б. оценен и утвержден.

- Результат каждого последующего этапа часто называется продуктом труда и обычно имеет форму некоторого документа (например, описания требований или описание проекта, программный код).
- Хотя набор документов, создаваемых в проекте зависит от его реализации, следующие документы обычно составляют обоснованный набор, который должен быть создан в каждом проекте:
  - спецификация требований;
  - план проекта;
  - документы по проектированию (архитектура системы, описание подсистем, детальный проект);
  - план тестирования и отчет о результатах ;
  - описание конечной программной реализации;
  - руководства по работе с ПО (администратора, пользователя).

The waterfall model

- Figure 2.1 The waterfall model



Преимущества модели водопада

- основным преимуществом является простота реализации;
- получение полной и согласованной документации на каждом этапе;
- легкость определения сроков и затрат на проект;
- простота администрирования ходом выполнения проекта (каждый этап завершается и создается его результат, некоторое количество денег передается потребителем разрабатывающей организации)

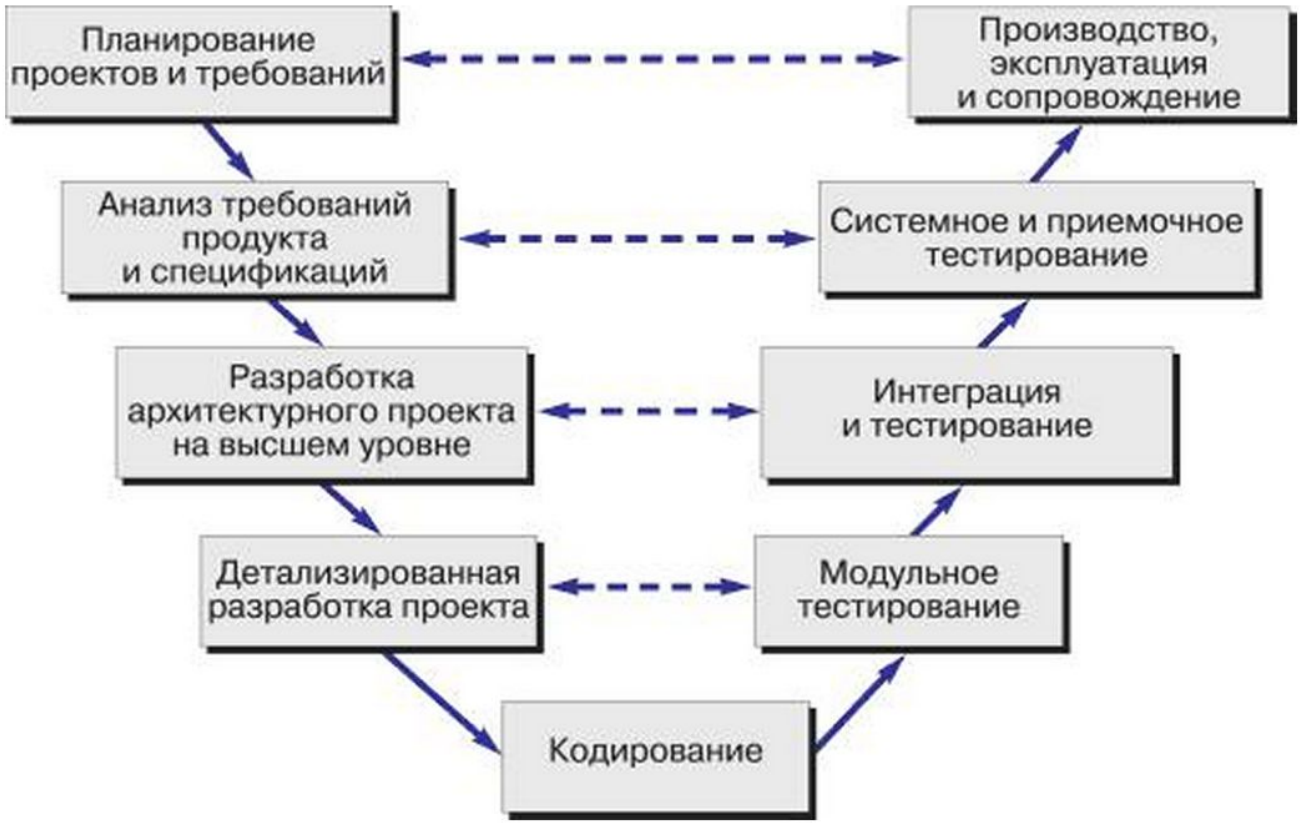
Недостатки

1. Предполагается, что ***требования к разрабатываемой системе м.б. неизменными (заморожены)*** до начала проектирования.
2. ***Замораживание требований обычно требует выбора технического обеспечения*** (т.к. это составляет часть спецификации требований).
3. Вызывает подход «большого взрыва» - полное ПО предоставляется за один раз, в конце разработки.
4. Поощряется «раздувание требований».
5. Данная модель является управляемой документами, что требует создание формального документа в конце каждого этапа.

- Несмотря на эти недостатки, модель водопада была одной из наиболее широко используемых моделей процесса разработки ПО.
- Данная модель хорошо подходит для рутинных проектов, в которых хорошо понятны требования.
- Таким образом, если:

1. разрабатывающая организация достаточно хорошо знакома с предметной областью,
2. требования к ПО являются достаточно ясными,

тогда модель водопада хорошо работает и


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

Выводы по модели водопадов

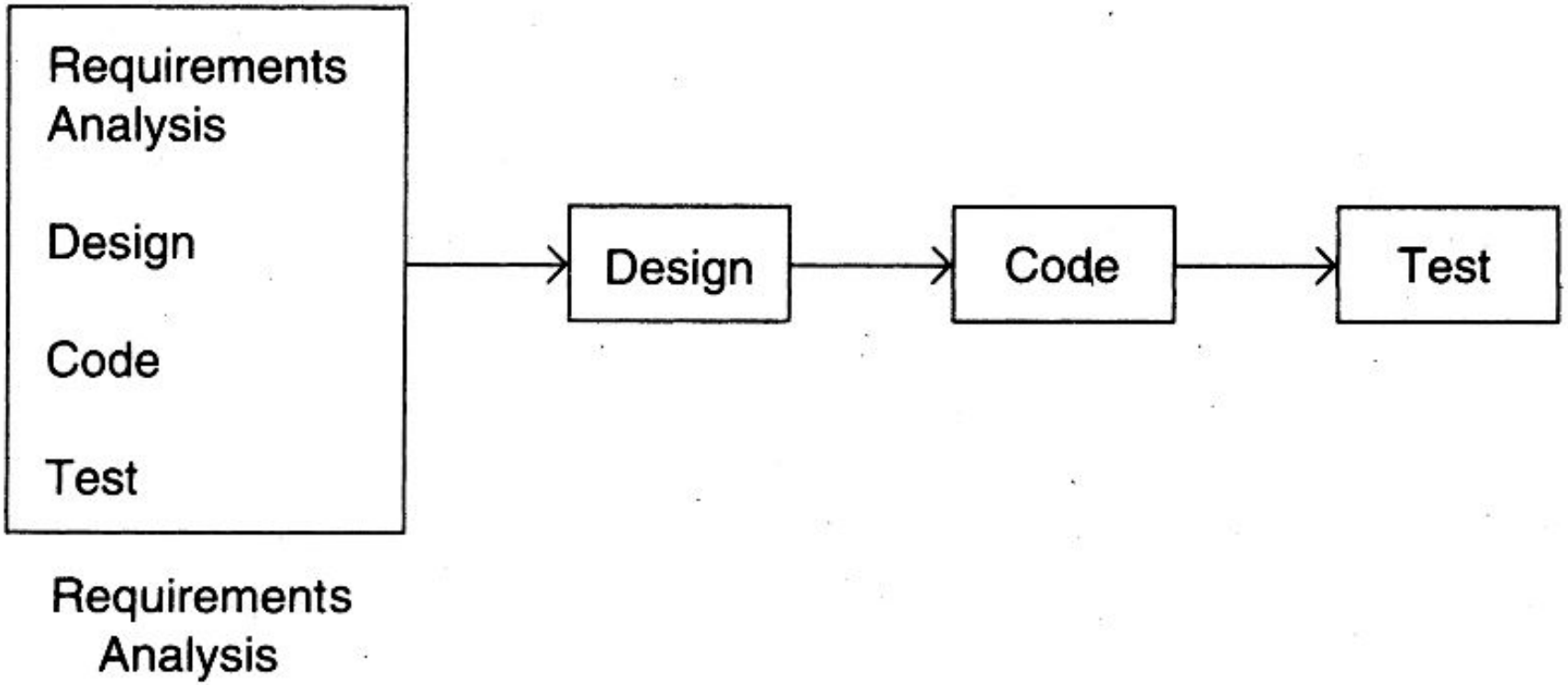
Достоинства	**Недостатки**	**Подходящие типы проектов**
Проста. Легкая для выполнения. Интуитивная и логически понятная. Легкая для выполнения договора.	Все или ничего – слишком рисковано. Рано замораживаются требования Может быть выбрано устаревшее оборудование/технология. Не позволяются изменения. Нет обратной связи с пользователями Поощряет раздувание требований.	Хорошо понятные проблемы. Краткосрочные проекты. Автоматизация существующих ручных систем.

2. Модель прототипирования

- Цель процесса разработки на основе прототипов заключается исправление первого ограничения модели водопадов.
- Основная идея – вместо замораживания требований, создавать, перед началом проектирования и кодирования, одноразовый прототип, который может помочь понять требования заказчика.
- Прототип сильно упрощенный вариант разрабатываемого ПО.
- Он разрабатывается на основе текущих известных требований.
- Очевидно, что разработка прототипа тоже проходит этапы проектирования, кодирования и тестирования, но каждый их этих этапов, но каждый из этих этапов выполняется не формально и не

- Прототипирование является привлекательной идеей для сложных и больших систем, для которых нет ручного процесса или существующей системы, которые могут помочь выявить требования.
- В таких ситуациях, предоставление заказчику возможности «поиграть» с прототипом позволяет выявить неоценимые и неосязаемые результаты, которые могут помочь определить требования к системе.
- Ситуации, в которых требуется пртотипирование:
  - для ***демонстрации осуществимости некоторого нового подхода***;
  - при разработке новых систем, когда не понятно, какие ограничения могут возникнуть, или какие алгоритмы могут быть разработаны для реализации требований.
- В обоих ситуациях риск, связанный с данными проектами м.б. уменьшен, за счет выполнения прототипирования.

Схема модели прототипирования



Процесс использования прототипа

- Процесс разработки с использованием прототипа обычно выполняется следующим образом:

1. Разработка прототипа обычно начинается, когда разработана предварительная версия документа с спецификациями требований.
2. После разработки прототипа, конечные пользователи и заказчику могут использовать и исследовать прототип.
   - Описываются и передаются разработчикам отзывы: что было сделано правильно; чего не хватает; что требуется добавить и т.п.
3. На основе отзывов выполняется доработка прототипа, а затем пользователям и заказчикам опять предоставляется возможность использовать данную ПС.
4. Такой цикл повторяется до тех пор, пока выгода от дальнейшего изменения прототипа превышает стоимость его доработки.
5. На основе такой обратной связи (отзывов), начальные требования модифицируются, чтобы получить окончательную спецификацию требований, которая затем используется разработки ПС производственного уровня.

Стоимость разработки прототипа

- Стоимость разработки прототипа должна быть очень низкой.
- Для этого в прототип включаются только те возможности, которые будут полезными для получения отзыва от пользователей.
- Способы уменьшения затрат на создание прототипа:
  - исключение модулей для реализации тех требований к ПО, которые уже хорошо понятны;
  - использование «быстрого и грязного» подхода, нацеленного на быструю разработку, а не на качество;
  - не используется: обработка исключений, аварийное восстановление, соответствие стандартам и форматам и т.п.;
  - создание только минимальной документации;
  - выполнение только минимально необходимого тестирования.
- С помощью таких способов уменьшения затрат на разработку прототипа, возможно довести стоимость создания прототипа до нескольких процентов общей

Достоинства прототипирования

1. Опыт полученный при разработке прототипа уменьшает стоимость разработки конечного ПО.
2. Получаются более стабильные требования, благодаря обратной связи с пользователями в требованиях будет меньше изменений.
3. Вероятно, что качество конечного ПО будет намного лучше, в связи с тем, что опыт полученный разработчики в ходе создания прототипа позволит улучшить проектирование, составление кода и выполнения тестирования.

Общий вывод

- Обычно прототипирование хорошо подходит для проектов, в которых трудно выявить требования и доверие к заявленным требованиям низкая.
- В тех проектах, у которых требования не совсем понятны в начале работы, использование модели процесса «прототипирование» может быть наиболее методом разработки ПО.
- Данный метод также является хорошим способом уменьшения некоторых рисков, связанных с проектом.

Выводы по прототипированию

Достоинства	**Недостатки**	**Подходящие типы проектов**
Помогает выявить требования. Уменьшает риски. Получается более стабильная конечная система.	Тяжелое начало. Возможны высокая стоимость и сжатые сроки. Поощряет раздувание требований. Затрудняет поздние изменения требований.	Системы с неопытными пользователями; или Области с неопределенными требованиями. Системы с большим кол-вом отчетов могут выиграть от прототипа пользовательского интерфейса.

3. Итеративная разработка

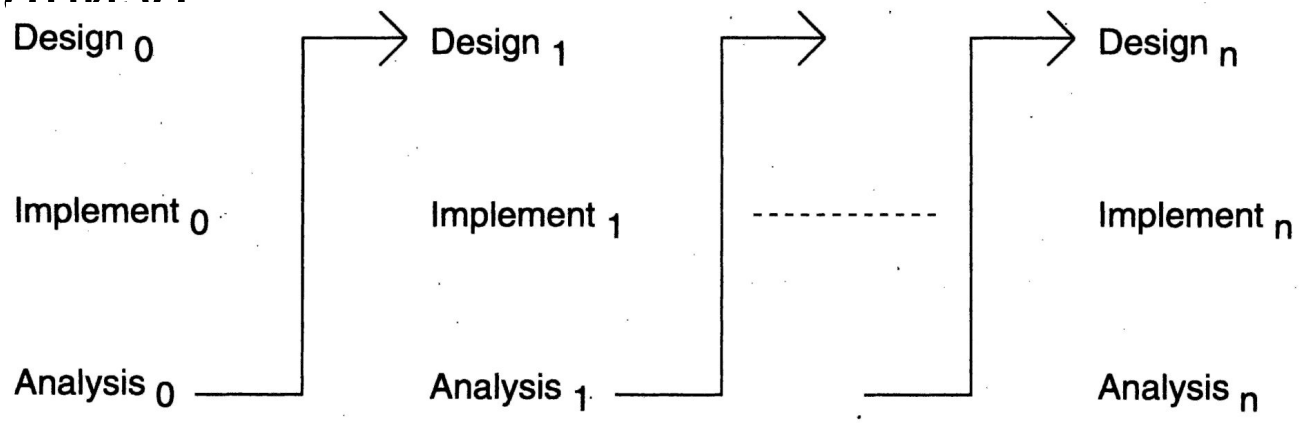
- Основная идея – пошаговая разработка ПО (постепенная, итеративная, инкрементальная).
- На каждом шаге к создаваемой ПС должны добавляться некоторые функциональные возможности, до тех пор, пока система не будет реализована полностью.
- На первом шаге данной модели создается простое начальное приложение, реализующее подмножество задач полного решения проблемы.
- Данное подмножество должно включать некоторые из основных задач рассматриваемой проблемы, которые легко понять и реализовать, и которые формируют полезную и простую в применении систему.

- Создается ***управляющий список***, который содержит упорядоченный набор задач, которые должны быть реализованы для получения конечного решения.
- Данный список проекта дает представление о том, как много этапов должно быть выполнено для получения окончательного варианта ПС.
- Управляющий список проекта руководит последовательностью шагов и содержит список всех задач, которые должны быть выполнены.

- На основе анализа, к задачам данного списка может относиться перепроектирование неправильно разработанных модулей или перепроектирование всей системы в целом.
- Однако перепроектирование системы обычно встречается только на начальных этапах.
- На более поздних этапах, проект должен стабилизироваться и вероятность его изменения должна уменьшаться.

Итеративная модель улучшения ПО

- Каждый шаг итеративной модели состоит из удаления следующей задачи из списка:
  1. проектирование реализации выбранной задачи
  2. кодирование и тестирование созданной реализации
  3. выполнение анализа частично разработанной системы, поученной после данного шага и обновление списка на основе результатов анализа



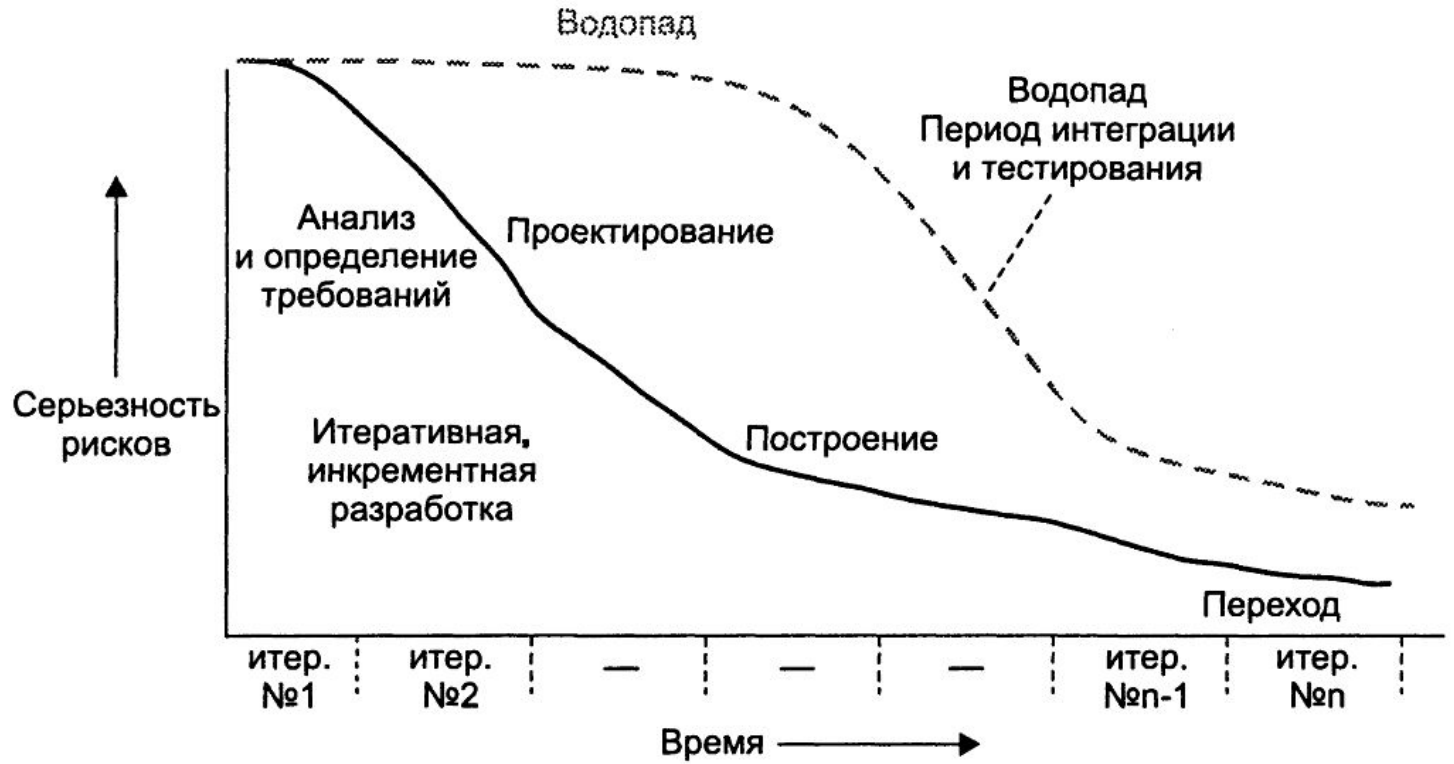
- ***Итеративная разработка в настоящее время является наиболее общим подходом для создания прикладных систем.***
- Данный подход м.б. планово-управляемым, гибким или, более часто, смесью этих подходов.
  - при планово-управляемым подходе требования к системе выявляются заранее.
  - при гибком подходе начальные итерации планируются, но разработка последующих итераций зависит от полученных результатов и приоритетов пользователей.

Достоинства

- Уменьшаются затраты на приспособление к изменившимся требованиям к ПО.
  - Уменьшается объем анализа и кол-во документации, которые должны быть переделаны.
- Легче получить отзыв потребителя на выполненные результаты разработки.
  - Потребитель может комментировать показываемое ему ПО и видеть, насколько много было разработано.
  - Потребителю трудно оценивать ход разработки на основе документов проектирования ПО.
- Возможно ускорить предоставление и развертывание полезного ПО у потребителя, даже если не вся требуемая функциональность в нем имеется.
- Отсутствует риска получить «все или ничего».

Сравнение рисков итеративной и водопадной моделей

- Серьезные риски при итеративной разработке определяются и уменьшаются раньше, чем при водопадной



Недостатки

- Очень долгое время отсутствует целостное понимание возможностей и ограничений проекта.
- При итерациях приходится отбрасывать часть сделанной ранее работы.
- Снижается добросовестность специалистов при выполнении работ, что психологически объяснимо, ведь над ними постоянно довлеет ощущение, что «всё равно всё можно будет переделать и улучшить позже».

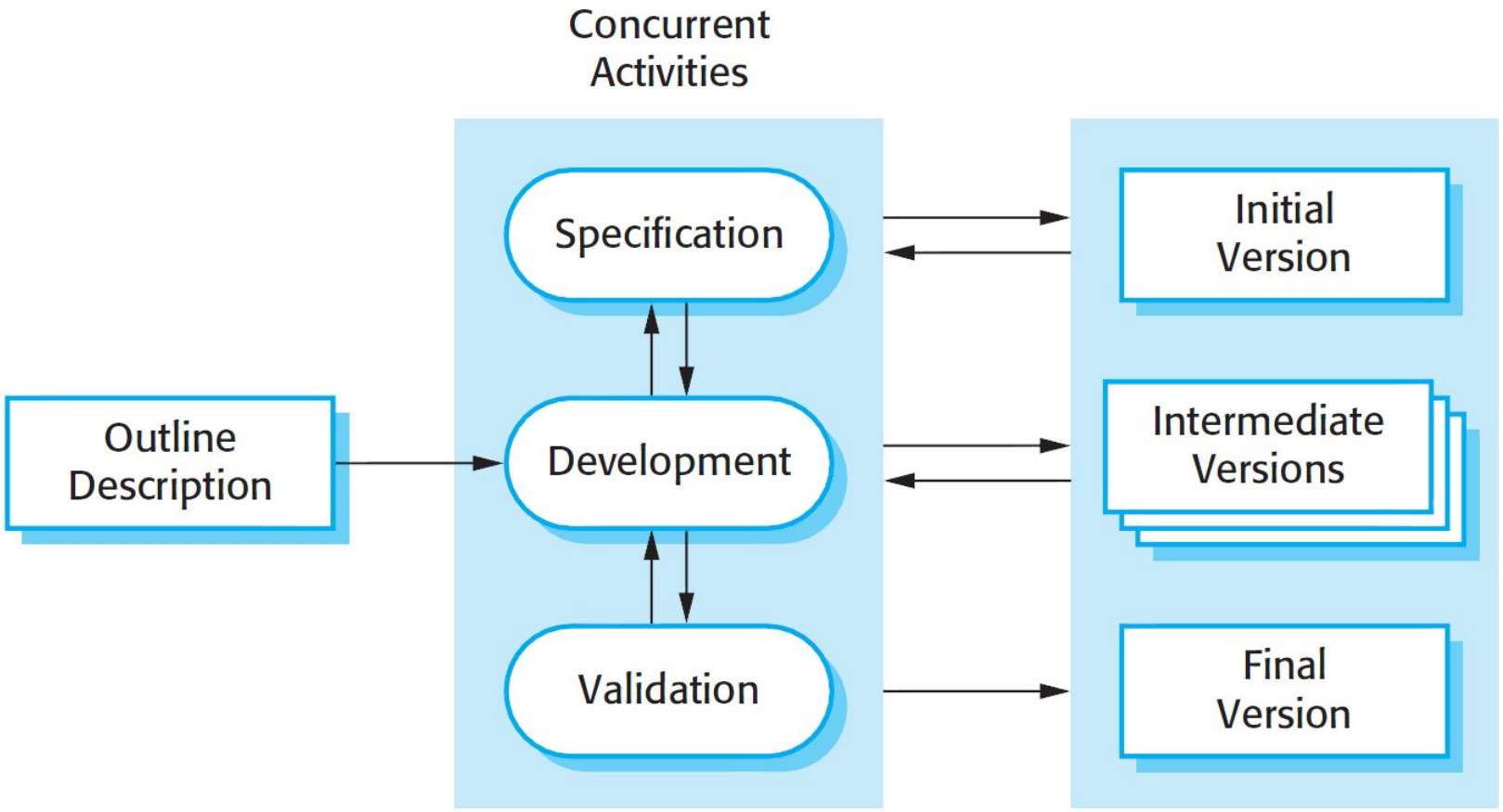
- Инкрементальную или гибкую модель разработки не всегда легко вводить или использовать в больших компаниях со стандартизированными инженерными процессами и в организациях, в которых разработка ПО обычно передается внешним исполнителям (аутсорсинг).
- Если разработка передается внешним исполнителям, то и клиент и исполнитель обычно хотят иметь полное описание того, что уже было сделано.

Основные трудности инкрементальной разработки

1. Проблемы менеджмента
   - Структура управления разработкой ПО в больших организациях создана для работы с такими моделями процесса разработки ПО, которые создают регулярно результаты, по которым можно оценить прогресс разработки.
   - Инкрементальные системы разработки меняются так быстро, что не выгодно по затратам создавать большое количество системной документации.
   - Менеджеры могут посчитать затруднительным использовать существующих сотрудников для выполнения инкрементальных процессов разработки, т.к. они не обладают таким опытом (умением).

Incremental development

- Figure 2.2 Incremental development



Причины популярности итеративной модели

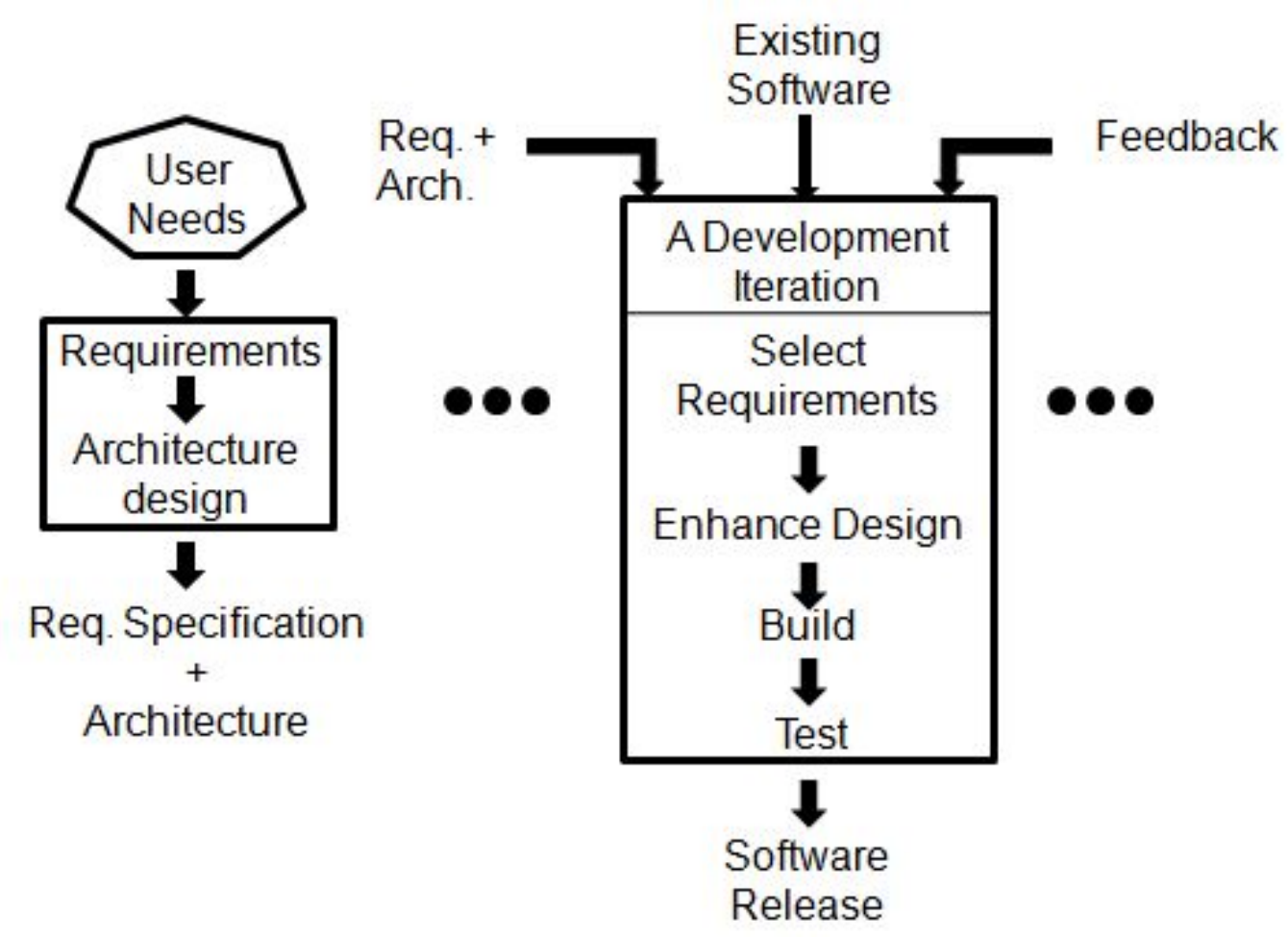
- В современном мире заказчик не хочет вкладывать деньги, если он не видит результаты от их вложения.
- Т.к. бизнес меняется быстро, то они никогда не знают «полностью» требования к ПО и им нужна возможность постоянно добавлять новые возможности к создаваемому ПО, чтобы оно соответствовало новым условиям.
- На каждой итерации создается работающая версия ПО, которая помогает разработать стабильные требования для следующей итерации.

Выводы по итеративному подходу

Достоинства	**Недостатки**	**Подходящие типы проектов**
Regular deliveries, leading to biz benefit Can accommodate changes naturally Allows user feedback Avoids req bloating Naturally prioritizes req Allows reasonable exit points Reduces risks	Overhead of planning each iteration Total cost may increase System arch and design may suffer Rework may increase	For businesses where time is imp; risk of long projects cannot be taken; req not known and evolve with time

Вариант итеративной модели

- **Инкрементальная модель** это вариант итеративной модели разработки



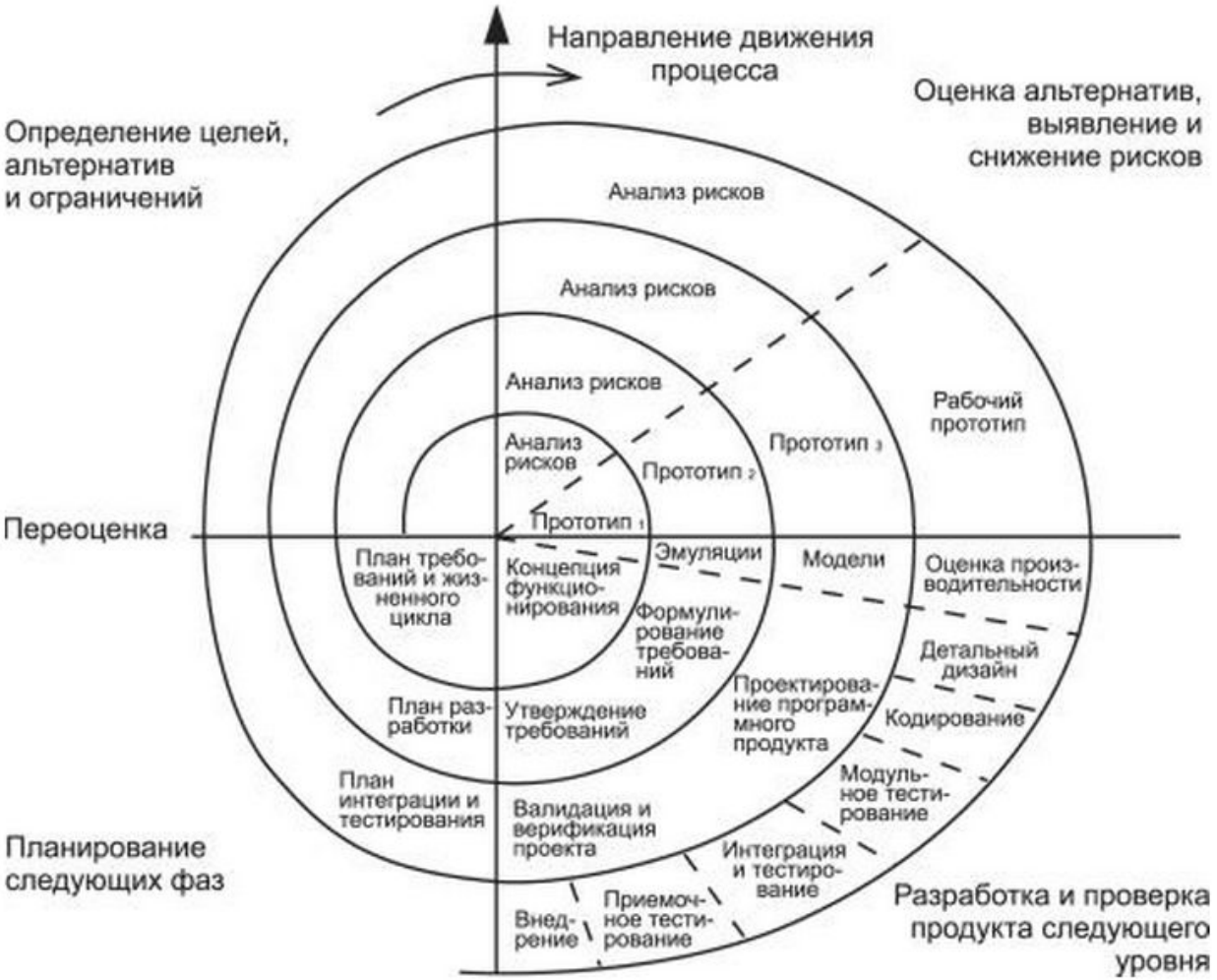
Достоинства данного варианта

- В большинстве случаев, требования к ПО заранее не известны.
- Доступно общее представление о ПС и на основе этого м.б разработана подходящая архитектура, которая остается относительно стабильной.
- В связи с этим есть надежда, что количество переделок в итерациях разработки будет уменьшаться.
- Одновременно, конечный результат будет предоставляться пользователю итеративно (постепенно), так что не будет риска получить «все или ничего».
- Т.к. поставка ПО выполняется постепенно, а планирование и выполнение каждой итерации выполняется отдельно, то пожелания пользователей м.б. включены в следующие итерации.
- Даже новые требования, которые были первоначально не выявлены, также могут быть включены в последующие итерации.

Итеративная модель

- *Рис. 4. Итеративная модель предлагает использование итераций на всех этапах жизненного цикла.*

Спиральная модель



Спиральная модель

1 — начальный сбор требований и планирование проекта; *2* — та же работа, но на основе рекомендаций заказчика; 3 — анализ риска на основе начальных требований; *4* — анализ риска на основе реакции заказчика; 5 — переход к комплексной системе; 6 — начальный макет системы; 7 — следующий уровень макета; *8* — сконструированная система; 9 — оценивание заказчиком

4. Унифицированный процесс разработки ПО (Rational Unified Process, RUP)

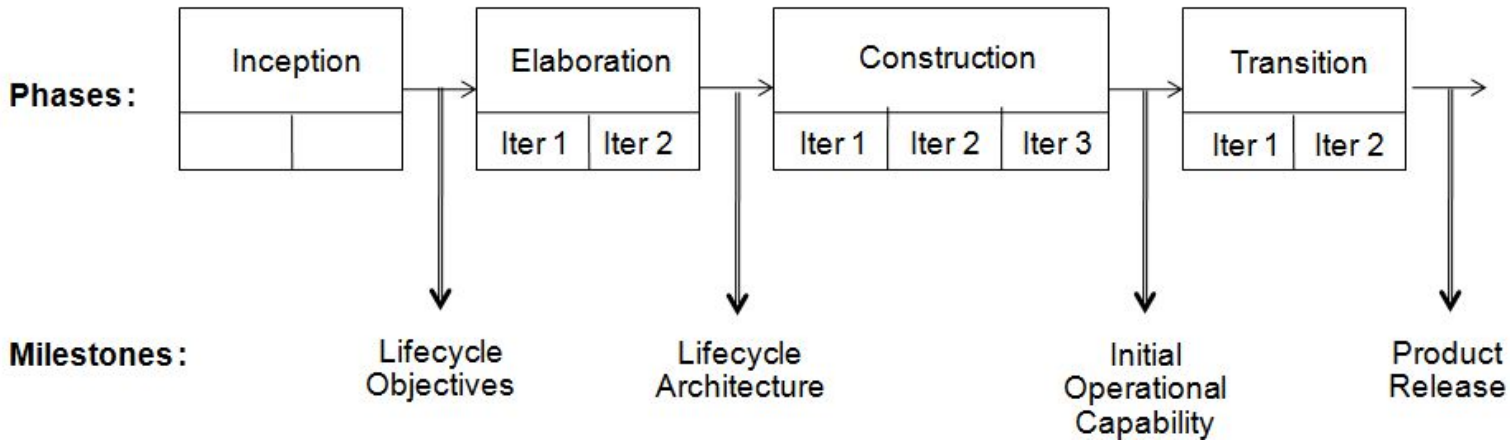
- Унифицированный процесс это вариант итеративного процесса, разработанный компанией Rational Software, (сейчас подразделение IBM).
- общая модель процесса, созданная для ОО разработки с использованием UML.
- Унифицированный процесс (UP) разрабатывается с 1967 года.
  - управляемым рисками и прецедентами (требованиями);
  - архитектуро-центричным;
  - итеративным и инкрементным.
- Это сложившийся открытый процесс разработки ПО от авторов UML.
- Унифицированный процесс компании Rational (RUP) – это коммерческое расширение UP.

- Процесс, направляемый вариантами использования
- Архитектуро-центрированный процесс
- Итеративный и инкрементный процесс

- Унифицированный процесс (UP) разрабатывается с 1967 года.
  - управляемым рисками и прецедентами (требованиями);
  - архитектуро-центричным;
  - итеративным и инкрементным.
- Это сложившийся открытый процесс разработки ПО от авторов UML.
- Унифицированный процесс компании Rational (RUP) – это коммерческое расширение UP.
- Он полностью совместим с UP, но более полный и детализированный.
- UP (и RUP) должны настраиваться под каждый конкретный проект путем добавления внутренних стандартов и др.

Схема модели RUP

- Программное обеспечение разрабатывается в течении 4 этапов (фаз):
  1. фаза анализа и планирования требований (начало) (inception)
  2. фаза проектирования (развития, уточнения) (elaboration)
  3. фаза построения (разработки) (construction)
  4. фаза внедрения (перехода)



- Обычно, каждая фаза выполняется в виде отдельного проекта, целью которого является дополнение дополнительных возможностей существующей системе (разработанной в предыдущем цикле).

Фазы унифицированного процесса разработки

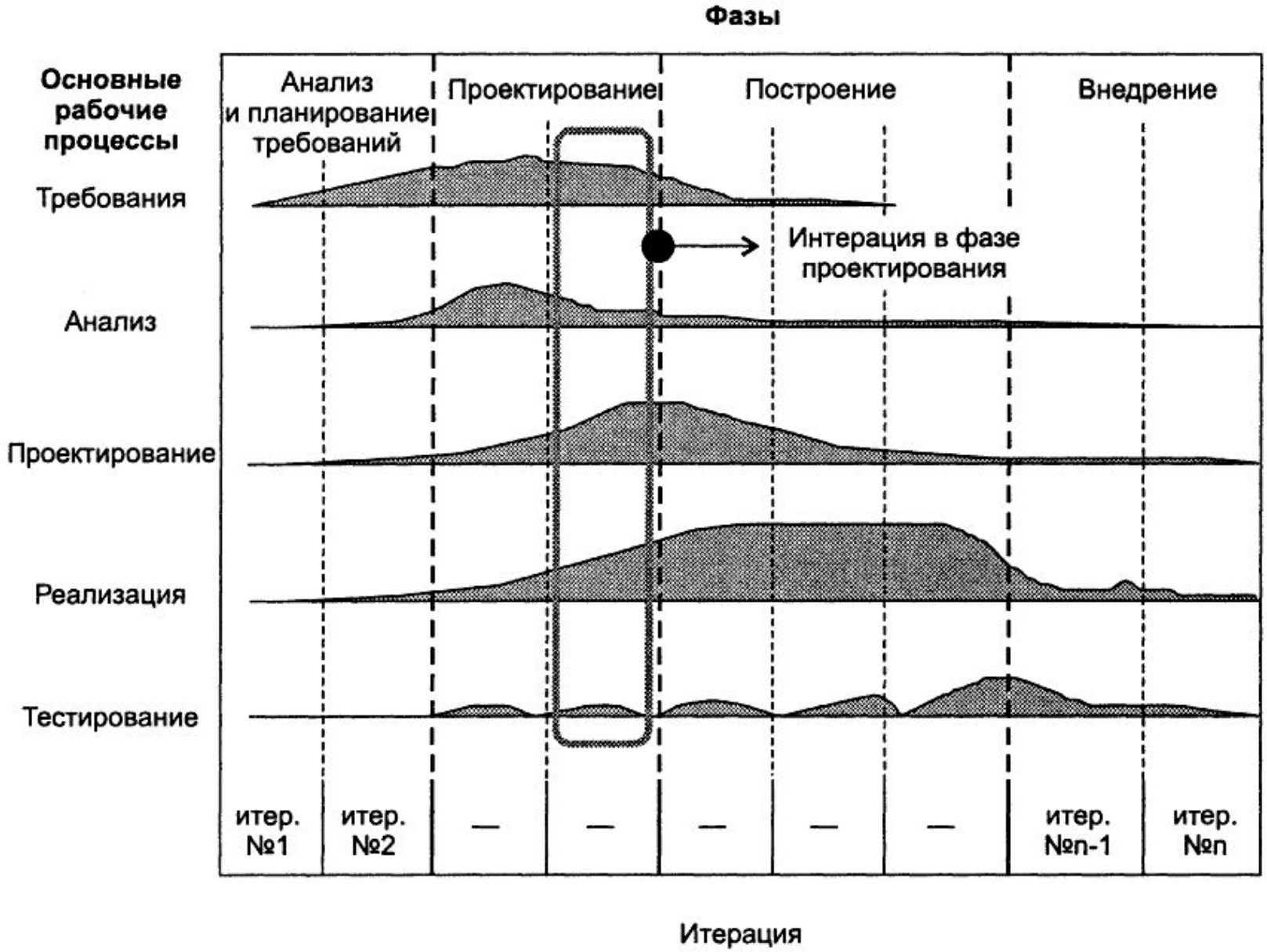
- У каждой фазы есть

1. цель,
2. основная деятельность с акцентом на одном или более рабочих потоках, и
3. контрольная точка.

Цели и результаты фаз разработки

1. **Начало** – проект сдвигается с «мертвой точки»:
   – Результат: определяются цели жизненного цикла;
2. **Проектирование** (уточнение) – развитие архитектуры системы:
   – Результат: определяется архитектура программной системы;
3. **Построение** – построение программного обеспечения:
   – Результат: разрабатывается базовая функциональность;
4. **Внедрение** – развертывание программного обеспечения в пользовательской среде:
   – Результат: выполняется выпуск продукта и его развертывание.

- В каждой фазе выполняется пять основных рабочих потоков:
  1. **определение требований** – выяснение того, что должна делать система;
  2. **анализ** – конкретизация и структурирование требований;
  3. **проектирование** – реализация требований в архитектуре системы (как система это делает);
  4. **реализация** – построение программного обеспечения;
  5. **тестирование** – проверяется, работает ли должным образом реализация.


Фазы
Основные рабочие процессы
Анализ и планирование требований
Проектирование
Построение
Внедрение
Требования
Интерация в фазе проектирования
Анализ
Проектирование
Реализация
Тестирование
итер. №1
итер. №2
итер. №n-1
итер. №n
Итерация

1. Фаза анализа и планирования требований (начало)

- Целью начального этапа является определение целей и масштаба проекта, а результатом данного этапа являются цели данной итерации (цикла).
- Полученный результат должен определять общее описание и высокоуровневые возможности созданной системы:
  - выгоды для бизнеса;
  - некоторые показательные варианты использования системы;
  - ключевые риски проекта;
  - базовый план проекта, включающий затраты и сроки.
- На основе этого принимается решение о том,

2. Фаза проектирования (развития)

- На этапе развития на основе детального анализа требований проектируется архитектура системы.
- Ожидается, что в конце данного этапа будет
  - выявлено и детально описано большинство требований;
  - разработана и описана архитектура системы, с учетом выявленных на раннем этапе технических рисков.
- Кроме этого, должен быть разработан высокоуровневый план проекта, описывающий оставшиеся этапы и итерации, и текущее понимание рисков.
- К концу данного этапа принимаются критически важные инженерные решения, касающиеся выбора технологий, архитектуры и т.п. и формируется детальное понимание данного проекта.

3. Фаза построения

- На этапе построения разрабатывается и тестируется ПО.
- Результатом данного этапа является переданный заказчику программный продукт, вместе с инструкциями (пользовательскими и др.).
- Успешное завершение данного этапа означает, что была достигнута веха начальных операционных возможностей.

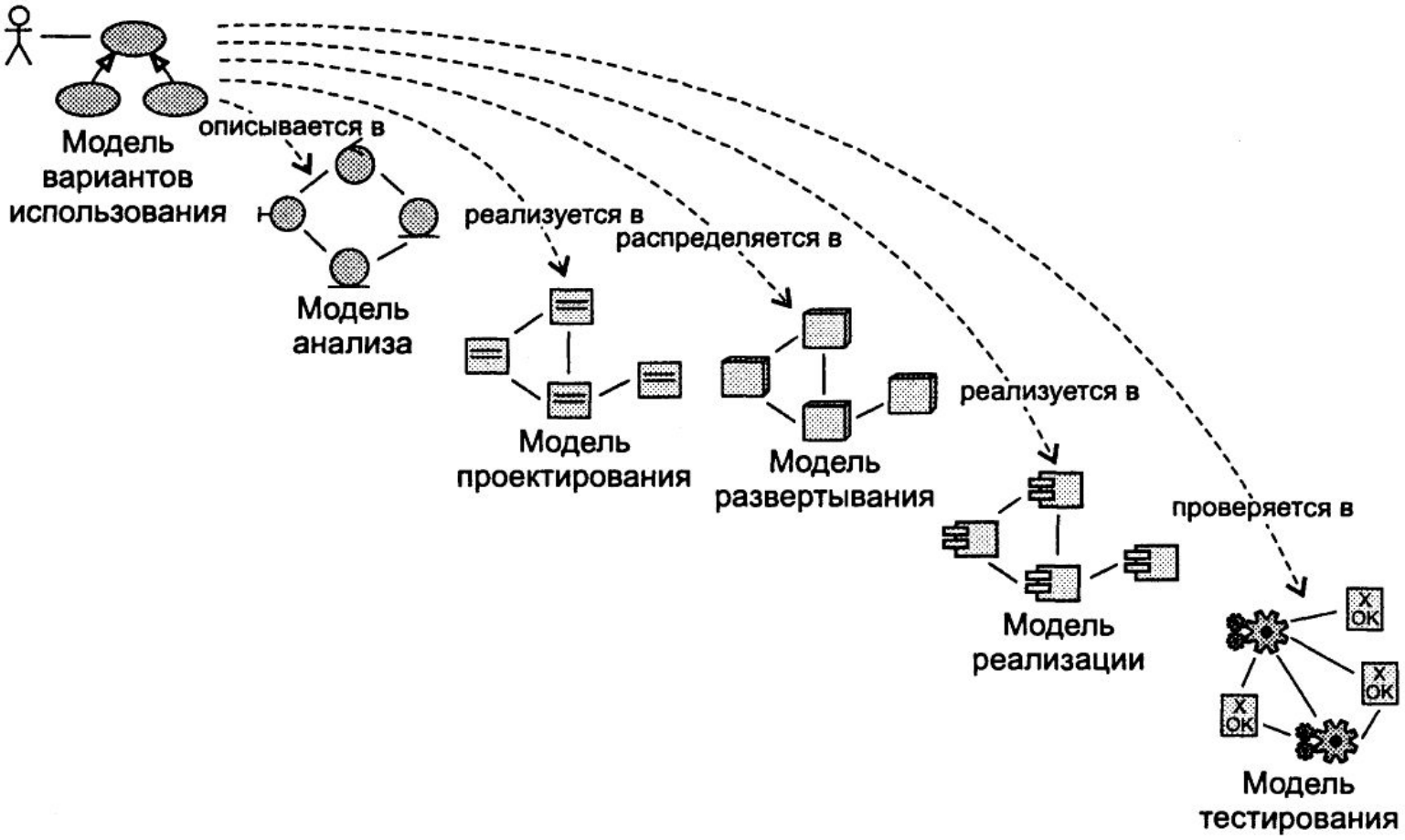
4. Фаза внедрения (перехода)

- Целью данной фазы перехода является перенос ПО из среду разработки в среду заказчика, где она должна работать.
- Это достаточно сложная задача, требующая
  - дополнительного тестирования,
  - преобразование старых данных заказчика для их использования в новой системе,
  - обучение персонала и
  - т.п.
- Успешное выполнение данной фазы приводит к достижению контрольной точки выпуска новой версии ПО.

Итерации выполнения фаз

- Хотя эти фазы выполняются последовательно, в рамках одной фазы могут выполняться несколько итераций.
- В результате каждой итерации выполняется предоставление внутренним или внешним потребителям некоторого хорошо определенного результата, который часто является частью конечной версии результата данной фазы.
- Обычно предполагается, что фаза построения будет включать несколько итераций, в каждой из которых создается работающая ПС.
- Результат каждой итерации оценивается пользователями и их отзывы используются в других итерациях.

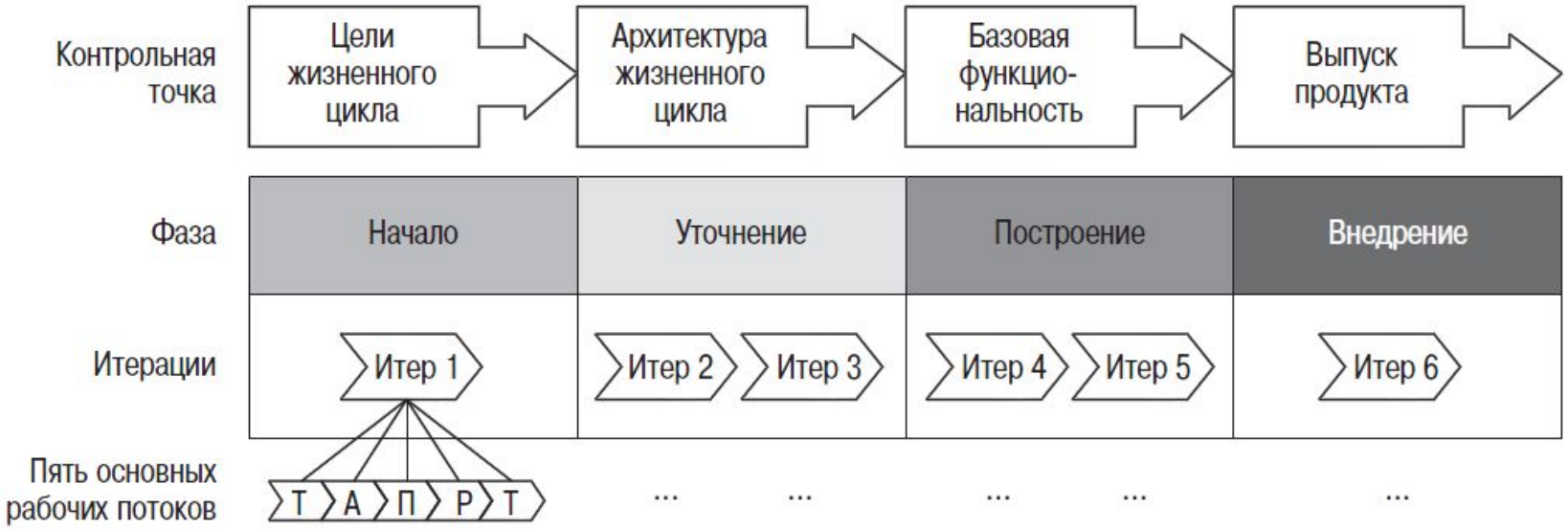
Модели Унифицированного процесса



Степень активности подпроцессов на разных этапах RUP

	Начало	Развитие	Построение	Внедрение
Требования	Высокая	Высокая	Низкая	Нет
Анализ и проектирование	Низкая	Высокая	Средняя	Нет
Реализация	Нет	Низкая	Высокая	Низкая
Тестирование	Нет	Низкая	Высокая	Средняя
Развертывание	Нет	Нет	Средняя	Высокая
Управление проектом	Средняя	Средняя	Средняя	Средняя
Управление конфигурирование	Низкая	Низкая	Высокая	Высокая

- Ключевым отличием RUP от других моделей заключается в том, что она отделяет фазы разработки от задач и позволяет множество таких подпроцессов выполняться в рамках фазы.
- В модели водопада (или итеративной модели, основанной на водопаде), этап процесса был связан с конкретной задачей, выполняемой некоторым процессом, например выявление требований, проектирование и т.п.
- В RUP такие задачи отделены от фаз (этапов), что позволяет , например, в ходе этапа построения выполнять задачу выявления требований.
- Т.е. это позволяет некоторые работы по выявлению требований выполнять даже при построении (создании) ПО, что не разрешено в модели водопада


Контрольная точка
Цели жизненного цикла
Архитектура жизненного цикла
Базовая функцио-нальность
Выпуск продукта
Фаза
Начало
Уточнение
Построение
Внедрение
Итерации
Итер 1
Итер 2
Итер 3
Итер 4
Итер 5
Итер 6
Пять основных рабочих потоков
Т
А
П
Р
Т
...
...
...
...
...

Фаза «Начало»

- Фаза Начало осуществляет инициирование проекта.
- Цель фазы: Начало – «сдвинуть проект с мертвой точки».
- Начало включает:
  - Обоснование выполнимости – может включать разработку технического прототипа с целью проверки правильности технологических решений или концептуального прототипа для проверки бизнес-требований.
  - Разработка экономического обоснования для демонстрации того, что проект обеспечит выраженную в количественном отношении коммерческую выгоду.
  - Определение основных требований для создания предметной области системы.
  - Выявление наиболее опасных рисков.

- Основными исполнителями в данной фазе являются:

1. руководитель проекта и
2. архитектор системы.

- Основное внимание обращено на определение требований и анализ.
- Однако если принято решение о создании технического или подтверждающего концепцию прототипа, может быть проведено некоторое проектирование и реализация.
- Тестирование обычно не применяется в данной фазе, поскольку единственными программными артефактами здесь являются прототипы, которые не будут больше нигде использоваться.

Цели фазы «проектирования»

1. Основная цель - создание исполняемой базовой версии архитектуры;
2. детализация оценки рисков;
3. определение атрибутов качества (скорости выявления дефектов, приемлемые плотности дефектов и т. д.);
4. выявление прецедентов, составляющих до 80% от функциональных требований;
5. создание подробного плана фазы **Построение**;
6. формулировка предложения, включающего ресурсы, время, оборудование, штат и стоимость.

- В фазе Проектирование основное внимание в каждом из основных рабочих потоков обращено на следующее:
  - определение требований – детализация предметной области системы и требований;
  - анализ – выяснение, что необходимо построить;
  - проектирование – создание стабильной архитектуры;
  - реализация – построение базовой версии архитектуры;
  - тестирование – тестирование базовой версии архитектуры.
- Основное внимание в фазе Проектирование направлено на рабочие потоки определения требований, анализа и проектирования.
- Реализация приобретает значение в конце фазы при создании исполняемой базовой версии архитектуры

Фаза «построение»

- Построение превращает исполняемую базовую версию архитектуры в за конченную рабочую систему.
- Цель фазы Построение –

1. завершить определение требований,
2. анализ и проектирование и
3. развить исполняемую базовую версию архитектуры, созданную в фазе Уточнение, в завершенную систему.

- Главная проблема Уточнения – *поддержание целостности архитектуры системы.*
- Очень часто при установлении сроков поставки и переходе к написанию кода начинается пренебрежение формальностями, что приводит к нарушению архитектурного представления, низкому качеству конечной системы и высоким затратам на обслуживание.

Фаза внедрение

- Внедрение направлено на развертывание законченной системы в сообществе пользователей.
- Внедрение начинается, когда завершено бета-тестирование и система окончательно развернута.
- Сюда относится устранение всех дефектов, обнаруженных при бетатестировании, и подготовка к массовому выпуску программного обеспечения на все пользовательские сайты.

Цели фазы внедрения

- Цели этой фазы можно обобщить следующим образом:

1. исправление дефектов;
2. подготовка пользовательских сайтов под новое программное обеспечение;
3. настройка работоспособности программного обеспечения на пользовательских сайтах;
4. изменение программного обеспечения в случае возникновения не предвиденных проблем;
5. создание руководств для пользователей и другой документации;
6. предоставление пользователям консультаций;
7. проведение послепроектного анализа.

Внедрение – на что обращено внимание

- Основное внимание концентрируется на рабочих потоках реализации и тестирования.
- Для исправления всех ошибок проектирования, выявленных при бета-тестировании, выполняется существенный объем проектирования.
- Надо стремиться к тому, чтобы в фазе Внедрение рабочие потоки определения требований и анализа оставались практически не задействованными

Выводы по методологии RUP

Достоинства	**Недостатки**	**Подходящие типы проектов**
Все преимущества итеративной модели. Предоставляет гибкий каркас для выполнения большого разнообразия проектов.	Для каждого проекта должен быть спроектирован его процесс.	Может быть применен к широкому набору проектов, т.к. это позволяет его гибкость.

5. Модель временных ящиков (Timeboxing Model)

- Для ускорения разработки может быть использована параллельность (одновременность) выполнения разных итераций.
- Новая итерация начинается прежде чем буде выпущена ПС созданная на данной итерации и сл-но разработка новой версии (варианта) происходит параллельно с разработкой текущей версии.
- Путем организация начала следующей итерации до того, как завершится предыдущая итерация возможно уменьшить среднее время предоставления ПП в итерациях.
- Однако, чтобы поддерживать параллельное выполнение, каждая итерация должна быть правильно структурирована и соответствующим образом должны быть сформированы команды исполнителей.

- Модель временных ящиков предлагает подход, соответствующий параллельной разработке.
- В модели временных ящиков основной единицей разработки является временной интервал (time box), имеющий фиксированную продолжительность.
- Т.к. эта продолжительность является фиксированной, то ключевым фактором выбора требований или возможностей, которые должны быть разработаны во временных ящиках, является то, что может поместиться в данный временной ящик.
- Это находится в противоречии с обычными итеративными подходами, в которых вначале выбирается функциональность, а затем определяется время ее программной реализации.

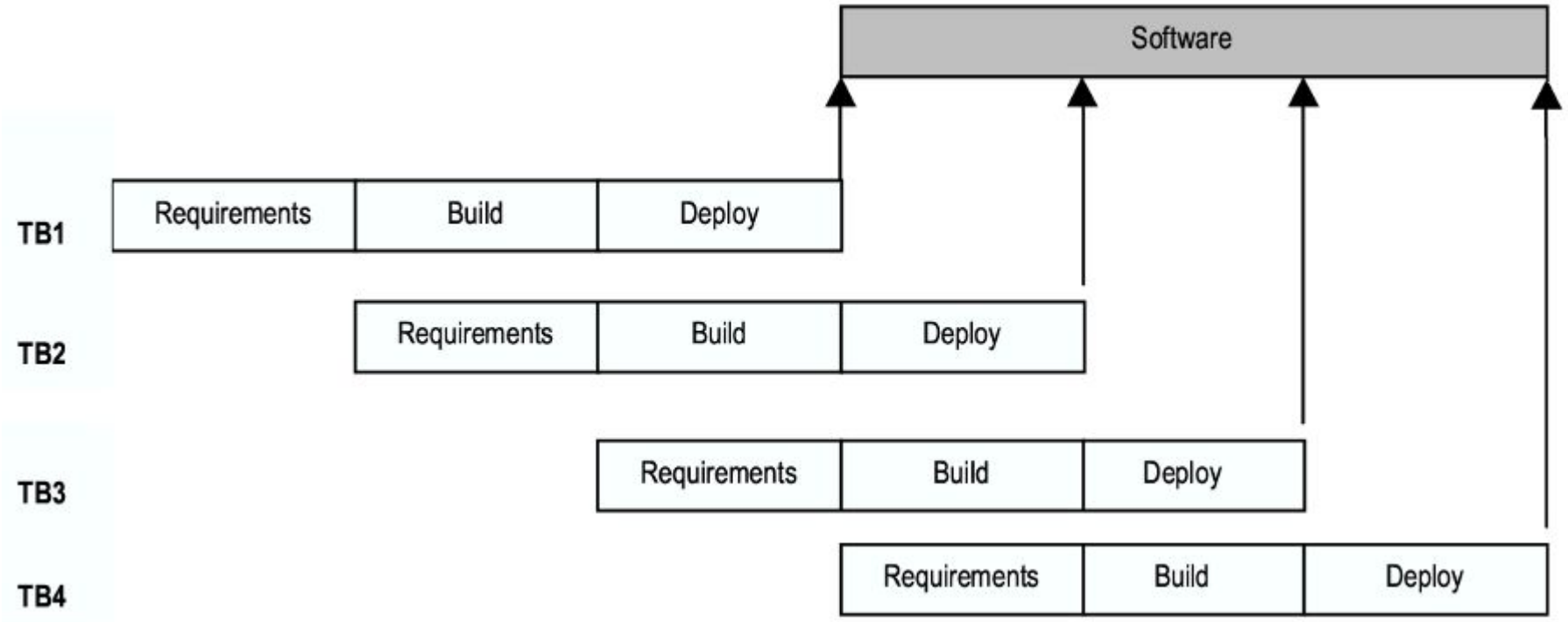
Временные интервалы

- Каждый временной ящик делится на последовательность этапов, как и в модели водопада.
- Каждый этап выполняет некоторую точно определенную задачу для данной итерации и создает точно определенный результат.
- Данная модель требует, чтобы продолжительность каждого этапа, т.е. время, требуемое для завершения задачи данного этапа, была примерно одинаковой.

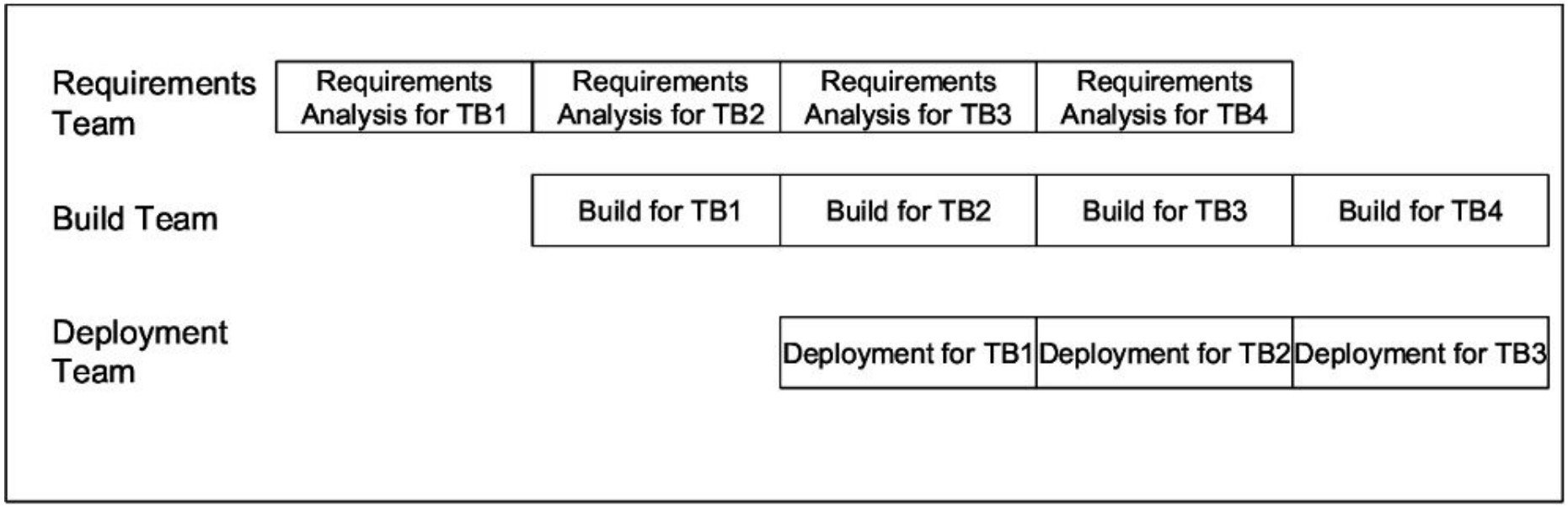
Специальные команды разработчиков

- Более того, данная модель требует, чтобы были организованы специальные команды для выполнения каждого этапа.
- Т.о., команда организованная для некоторого этапа выполняет только задачи данного этапа – задачи для других этапов выполняются другие специальные команды.
- Это очень сильно отличается от других итеративных моделей, в которых неявно предполагается, что одна и та же команда выполняет все различные задачи проекта или итерации.
- Если имеются итераций во временных ящиках, разделенных на этапы равной продолжительности, и специальные команд, которые могут выполнять такие этапы, то появляется возможность организации конвейерной обработки (pipelining).

Схема модели временных ящиков



Задачи разных команд



Область применения

Выводы по модели временных ящиков (Timeboxing)

Достоинства	**Недостатки**	**Подходящие типы проектов**
Такие же, как и у итеративной модели Планирование итераций делать несколько проще Очень короткое время предоставления ПО.	Управление проектом становится более сложным. Размер команды увеличивается. Усложнения (упущения) могут вести к потерям.	Когда очень важно короткое время разработки ПО. Когда существует гибкость группировки возможностей ПО. Стабильная архитектура.

6. Гибкие процессы разработки ПО

- набор подходов к разработке ПО, использующих
  - итеративную разработку;
  - динамическое формирование требований;
  - обеспечение их реализации путем постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля.
- начали создаваться с начала 90-х годов в ответ на требование детального документирования и бюрократические процессы других подхода (в особенности модели водопада).
- это легковесные подходы в отличие от тяжеловесных подходов (водопад, RUP и т.п.).

Гибкая процессы разработки (*agile software development*)

- Набор подходов к разработке программного обеспечения, ориентированных на
  - использование итеративной разработки и
  - динамическое формирование требований и
  - обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля.
- Существует несколько методик, относящихся к классу гибких методологий разработки, в частности, известны как гибкие методики
  - экстремальное программирование,
  - DSDM,
  - Scrum.

Принципы гибкой разработки

Гибкие подходы основываются на следующих базовых принципах [www.extremeprogramming.org]:

1. Работающее ПО является основным средством измерения развития проекта.

2. В таком случае, для развития проекта требуется, чтобы ПО разрабатывалось и предоставлялось быстро путем добавления небольших улучшений.

3. Даже поздние изменения в требованиях должны быть приняты (модель разработки на основе небольших приращений функциональности помогает их принимать).

4. Личное общение является более предпочтительным, чем обмен документами.

5. Непрерывная обратная связь с потребителем и вовлечение потребителя в разработку является необходимым для создания высококачественного ПО.

6. Простое проектирование, которое развивается и улучшается со временем, является лучшим подходом в сравнении с выполнением заранее детального проектирования, которое учитывает все возможные сценарии использования создаваемого ПО.
7. Сроки передачи готового ПО определяются квалифицированными командами талантливых разработчиков (а не путем распоряжений).

Разновидности гибких подходов

- XP,
- Agile,
- Lean,
- Scrum,
- Kanban,
- Theory of Constraints,
- System Thinking,
- Flow-Based Product Development
- и т.д.

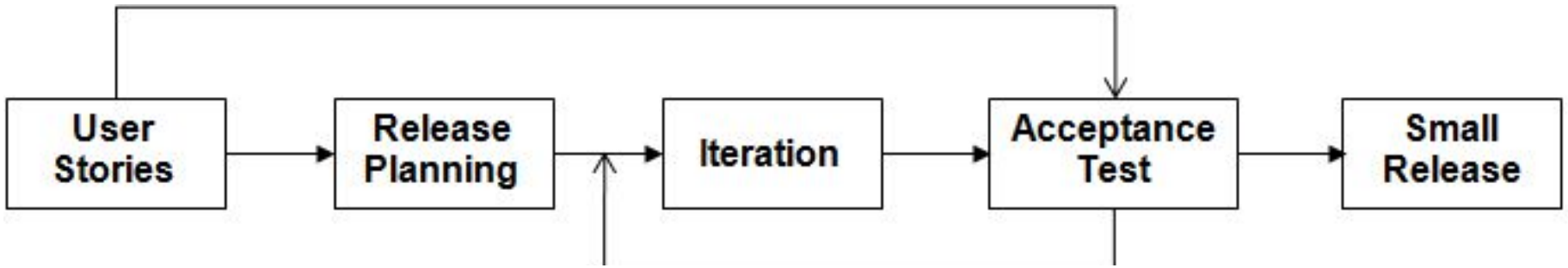
Экстремальное программирование (eXtreme Programming, XP)

- один из популярных и широко используемых подходов гибкой разработки ПО;
- предполагает, что изменения неизбежны и вместо борьбы с ними, разработка должна быть готова быстро их реализовать.
- считается, что для приспособления к изменениям, процесс разработки должен быть ***облегченным*** и

- В XP ПО разрабатывается итеративно и не создается детальная и многочисленной документации, которую трудно поддерживать.
- Основное внимание личному общению, простоте и обратной связи, для гарантирования того, что желаемые изменения быстро и правильно отражались в программах.

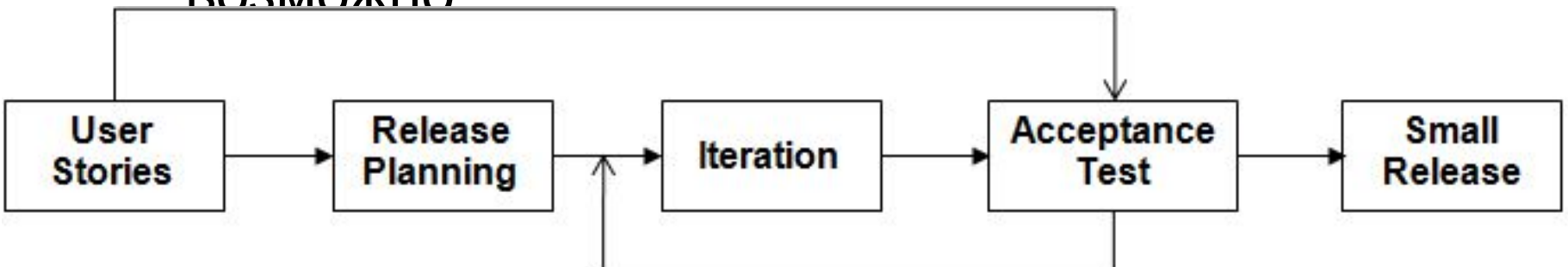
Схема XP модели процесса

- Проект начинается с «историй пользователей», которые являются короткими (в несколько предложения) описаний сценариев, которые потребители и пользователи хотели бы, чтобы поддерживало разрабатываемое ПО.

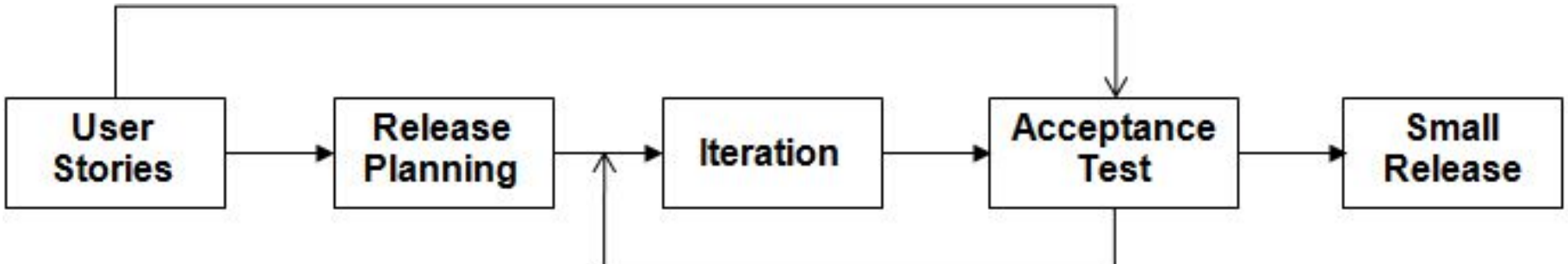


Истории пользователей

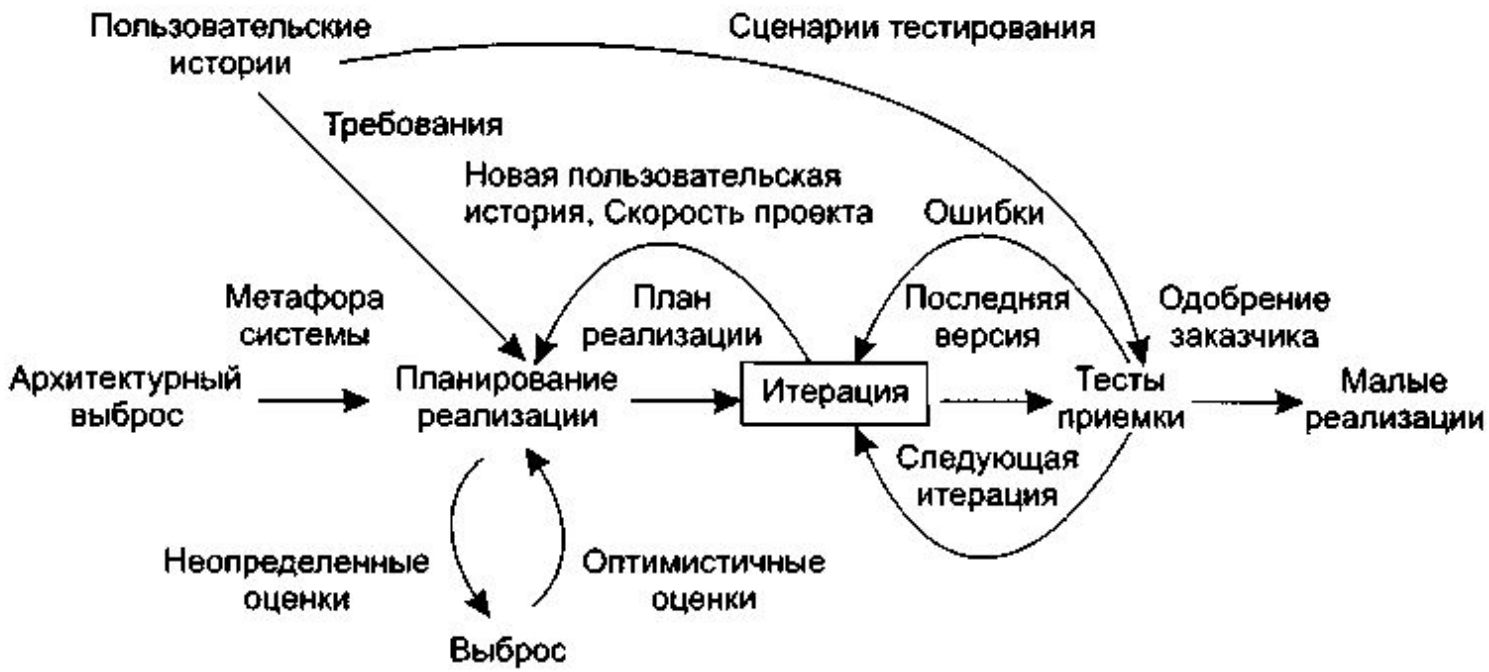
- отличаются от традиционных спецификаций (детальных описаний) требований в основном уровнем детальности:
  - не содержат детальные требования, которые будут определены только тогда, когда данная история будет реализована;
  - разрешается подробные требования предоставлять так поздно, настолько это возможно



- Каждая история записывается на отдельной карточке, так чтобы их можно было просто группировать.
- Назначенная команда разработчиков оценивает, как долго потребуется реализовать историю пользователя.
- Данные оценки являются грубыми и обычно задаются в неделях.

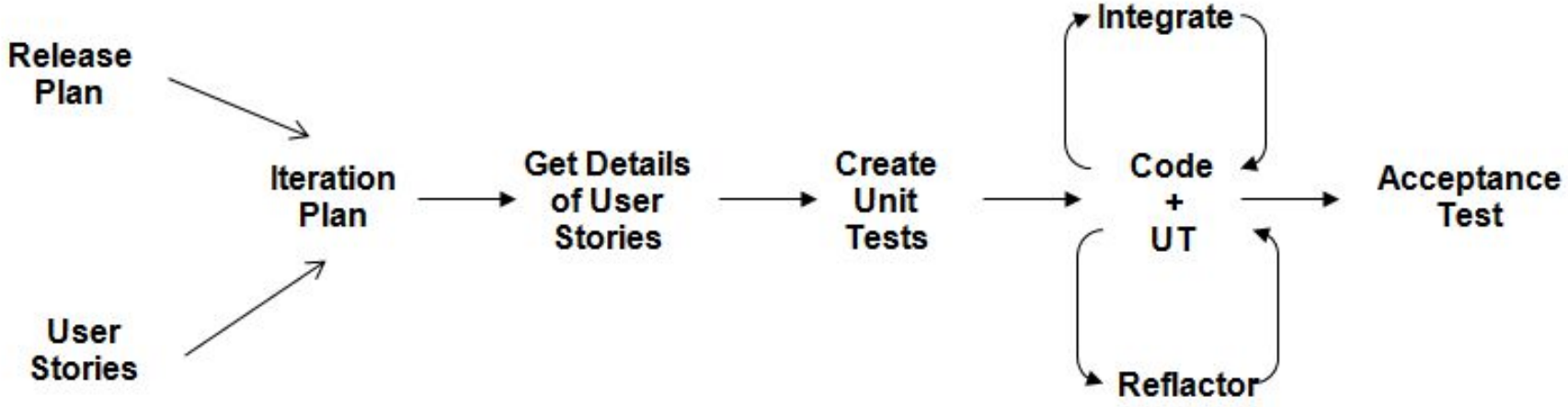


- С помощью таких оценок и историй выполняется планирование выпуска версий ПО, которое определяет какие истории должны быть реализованы в каких вариантах и даты их окончания.
- Поощряется частый выпуск небольших версий ПО, а для создания версий используются итерации.
- На основе историй также разрабатываются, ***приемочные*** тесты, которые используются для тестирования версий ПО перед их выпуском.


Пользовательские истории
Сценарии тестирования
Требования
Новая пользовательская история, Скорость проекта
Ошибки
Метафора системы
План реализации
Последняя версия
Одобрение заказчика
Архитектурный выброс
Планирование реализации
Итерация
Тесты приемки
Малые реализации
Следующая итерация
Неопределенные оценки
Оптимистичные оценки
Выброс

Итерации разработки

- Разработка выполняется в результате нескольких итераций. Каждая итерация продолжается несколько недель.
- Итерация начинается с планирования итерации, в ходе которого выбираются истории пользователей, которые должны быть реализованы на данной итерации.
- Для выполнения разработки определяются детали выбранных историй.
- Создаются тесты.
- Выполняется кодирование и тестирование



Особенности разработки в итерациях

1. Рассчитывается, что разработка выполняется парами программистов (т.н. парное программирование).
2. Предполагается, что до реального написания кода программных единиц создаются автоматизированные тесты, а затем код должен составляться так, что проходить эти тесты.
3. Используется разработка управляемая тестами (test-driven development), которая отличается от обычного подхода, когда сперва составляют код, а затем думают о том, как его

4. Поощряется создание простых решений и их дальнейшее изменение.
   - Ожидается, что проект решения разработанный заранее может в некоторый момент времени стать не подходящим для дальнейшей разработки.
5. Поощряется частая интеграция разных программных единиц.

Дополнительные правила выполнения итерации

- В XP имеется много других правил, таких, как:

1. Распределение прав между программистами и потребителями.
2. Взаимодействие между членами команда и использование метафор.
3. Доверие и открытость для всех заинтересованных лиц.
4. Коллективное владение кодом, при котором любая пара может изменить

5. Командное управление
6. Построение быстрых пробных решений, для апробации трудных с технических и архитектурных вопросов, для исследования некоторого подхода, способа ликвидации ошибок.
7. устанавливаются правила,
   - как проводить собрания разработчиков.
   - как должен начинаться день разработчиков.

- Выбор разных правил при выполнении текущей итерации определяется достигнутыми результатами предыдущей итерации.

Вариант гибкого процесса - SCRUM

- Scrum (скрам) предоставляет эмпирический подход к разработке ПО.
  - методология управления разработкой информационных систем для гибкой разработки программного обеспечения.
- Этот процесс является быстрым, адаптивным, может самонастриваться и отличается от последовательного (водопадного) подхода.
- Scrum основан на повторяющихся циклах, это делает его более гибким и

- В основе лежат короткие ежедневные встречи — Scrum и циклические 30-дневные встречи, называемые Sprint.
- Во время ежедневного Scrum’а каждому члену команды задаются только 3 вопроса:
  - Что было сделано с последнего Scrum’а?
  - Что вы будете делать между этой и следующий встречей Scrum?
  - Что мешает вам выполнять работу?

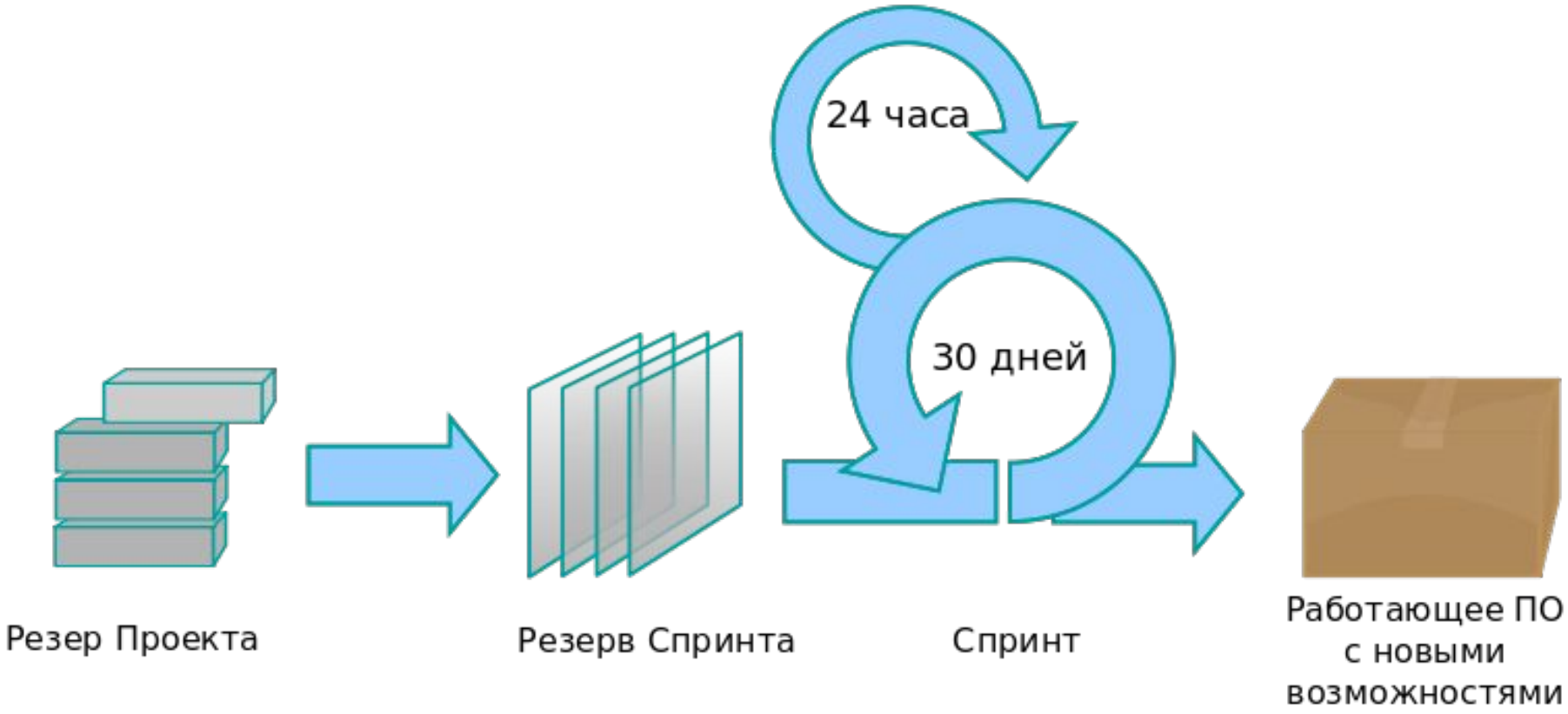
- Скрам (Scrum) — это набор принципов позволяющих в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет.
- Возможности ПО к реализации в очередном спринте 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.

Главные принципы Scrum

1. Индивидуализм и взаимодействие важнее строгих процессов и методов.
2. Работающее ПО важнее сложной документации.
3. Взаимодействие с заказчиками важнее контрактных договоренностей.
4. Готовность изменить важнее следованию плану.

- Scrum, как и другие способы быстрой разработки ПО лежит вне того представления, что главной задачей является делать все возможное для следования плану.
- Scrum — это искусство возможного.
- Девизом Scrum’a можно считать - следование здравому смыслу.

Скрам процессы



Спринт (sprint)

- Спринт — итерация в скрам, в ходе которой создаётся функциональный рост программного обеспечения.
- Жёстко фиксирован по времени.
- Длительность одного спринта от 2 до 4 недель.
  - в отдельных случаях длительность спринта должна быть не более 6 недель.
- 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.

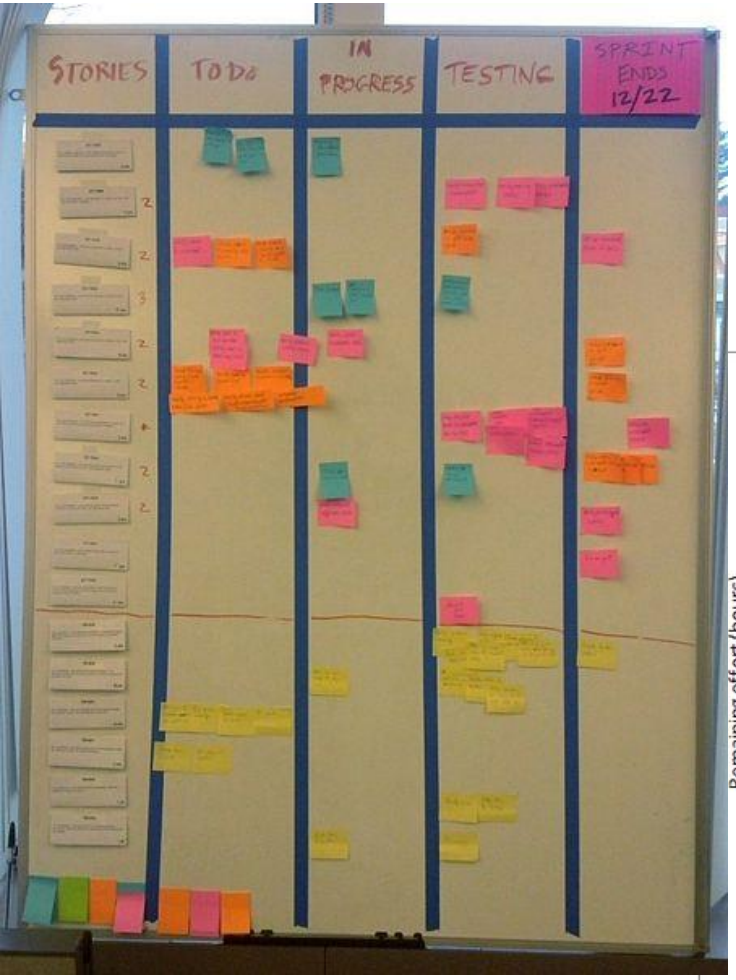
- Разные команды подбирают длину спринта согласно специфике своей работы, составу команд и требований, часто методом проб и ошибок.
- Для оценки объема работ в спринте можно использовать предварительную оценку, измеряемую в очках истории, где 1 очко истории приблизительно равно 1 человеко-дню.
- Предварительная оценка фиксируется в резерве проекта.
- На протяжении спринта никто не имеет права менять список требований к работе, внесенном в резерв проекта.

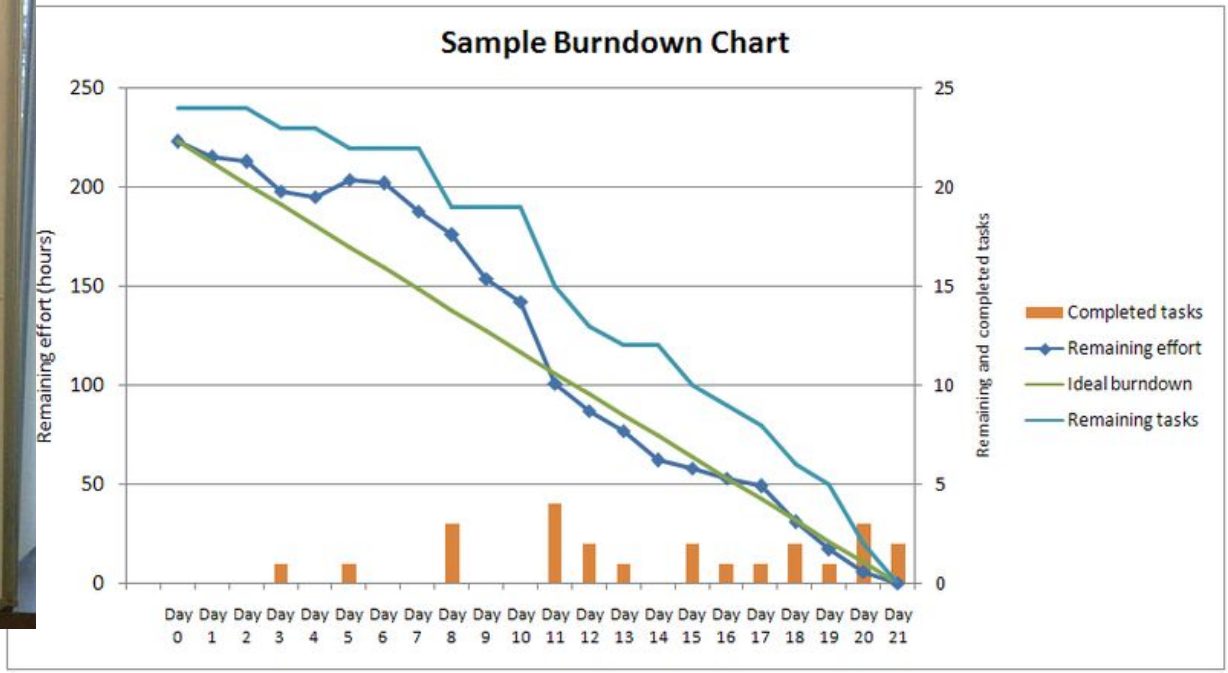
- **Резерв Проекта (Product backlog)** — это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации.
  - Элементы этого списка называются «пожеланиями пользователя» (*user story*) или элементами резерва (*backlog items*).
  - **Резерв проекта** открыт для редактирования для всех участников скрам процесса.
- **Резерв спринта (Sprint backlog)** — содержит функциональность, выбранную **владельцем проекта** из **резерва проекта**.
  - Все функции разбиты по задачам, каждая из которых оценивается **скрам-командой**.
  - Каждый день команда оценивает объем работы, который нужно проделать для завершения спринта.

Диаграмма сгорания задач (Burndown chart)

- Диаграмма, показывающая количество сделанной и оставшейся работы.
  - Обновляется ежедневно с тем, чтобы в простой форме показать подвижки в работе над спринтом.
  - График должен быть общедоступен.
- Существуют разные виды диаграммы:
  - диаграмма сгорания работ для спринта – показывает, сколько уже задач сделано и сколько ещё остаётся сделать в текущем спринте.
  - диаграмма сгорания работ для выпуска проекта – показывает, сколько уже задач сделано и сколько ещё остаётся сделать до выпуска продукта (обычно строится на базе нескольких спринтов).

Примеры документов





Роли в скрам-процессе

- По методике Scrum в производственном процессе есть определенные роли, разбитые на 2 группы: «поросята (свиньи)» и «циплята (куры)».
  - *свиньи* создают продукт,
  - *куры* заинтересованы, но не настолько — ведь им всё равно, будет ли проект удачным или нет, на них это мало отразится.
- Требования, пожелания, идеи и влияние *кур* принимаются во внимание, но им не разрешают непосредственно включаться в ход скрам-проекта.

«Свиньи»

- **скрам-мастер (ScrumMaster)** – проводит совещания (Scrum meetings) следит за соблюдением всех принципов скрам, 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 проекта (Product Owner)** – представляет интересы конечных пользователей и других заинтересованных в продукте сторон.
- **скрам-команда (Scrum Team)** – кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: тестировщиков, архитекторов, аналитиков, программистов и т. д.
  - Размер команды в идеале составляет 7±2 человека.
  - 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.

Скрам-команда (Scrum Team)

- кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: тестировщиков, архитекторов, аналитиков, программистов и т. д.
- Размер команды в идеале составляет 7±2 человека.
- 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.

Scrum команда

- Команда — главная движущая сила проекта, это источник продуктивности и креативности.
- Команда, которая будет участвовать в ежедневных Scrum’ах должна состоять из 7 плюс-минус 2 человека.
- Большее число не рекомендуется, лучше разделить на несколько команд.
- Scrum Master (или, более понятно – Project manager) ведет эту встречу.
- Члены команды называются «поросята" (pigs) и только они имеют право говорить.

- На встречах могут присутствовать молчаливые посетители — "циплята" (*chickens*), которые такого права лишены.
- Scrum встречи должны быть короткими, опоздания не приветствуются.
- В команде не должны быть только разработчики.
- Команда должна включать в себя также и тестеров, дизайнеров, а также других заинтересованных лиц.
- Желательно, если рабочие места всей команды находятся в одной комнате, либо, как минимум в соседних.

«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*)**

Планирование спринта (Sprint Planning Meeting)

- Происходит в начале новой итерации Спринта.
- Из резерва проекта выбираются задачи, обязательства по выполнению которых за спринт принимает на себя команда;
- На основе выбранных задач создается резерв спринт. Каждая задача оценивается в идеальных человеко-часах;
- Решение задачи не должно занимать более 12 часов или одного дня.
  - При необходимости задача разбивается на подзадачи;

- Обсуждается и определяется, каким образом будет реализован этот объём работ;
- Продолжительность совещания ограничено сверху 4-8 часами в зависимости от продолжительности итерации, опыта команды и т. п.
  - (первая часть совещания) Участвует **владелец проекта** и **скрам команда**: выбирают задачи из резерва продукта;
  - (вторая часть совещания) Участвует только команда: обсуждают технические детали реализации, наполняют резерв спринта.

Ежедневное совещание (Daily Scrum meeting)

- начинается точно вовремя;
- все могут наблюдать, но только «свиньи» говорят;
- длится не более 15 минут;
- проводится в одном и том же месте в течение спринта.
- В течение совещания каждый член команды отвечает на 3 вопроса:
  - Что сделано с момента предыдущего ежедневного совещания?
- Что будет сделано с момента текущего совещания до следующего?
  - Какие проблемы мешают достижению целей спринта? (Над решением этих проблем работает *скрам мастер*. Обычно это решение проходит за рамками ежедневного совещания и в составе лиц, непосредственно затронутых данным препятствием.)

Ежедневное совещание

Скрам над скрамом (Scrum of Scrums)

- Проводится после ежедневного скрам совещания.
  - Позволяет нескольким скрам командам обсуждать работу, фокусируясь на общих областях и взаимной интеграции.
- Повестка та же, что и на ежедневном скрам совещании плюс следующие вопросы:
  - Что каждая команда сделала с момента предыдущего ежедневного совещания?
  - Что каждая команда сделает к следующему ежедневному совещанию
  - Есть ли проблемы, мешающие или замедляющие работу каждой команды?
  - Нужно ли другой команде сделать что-то из задач вашей команды?

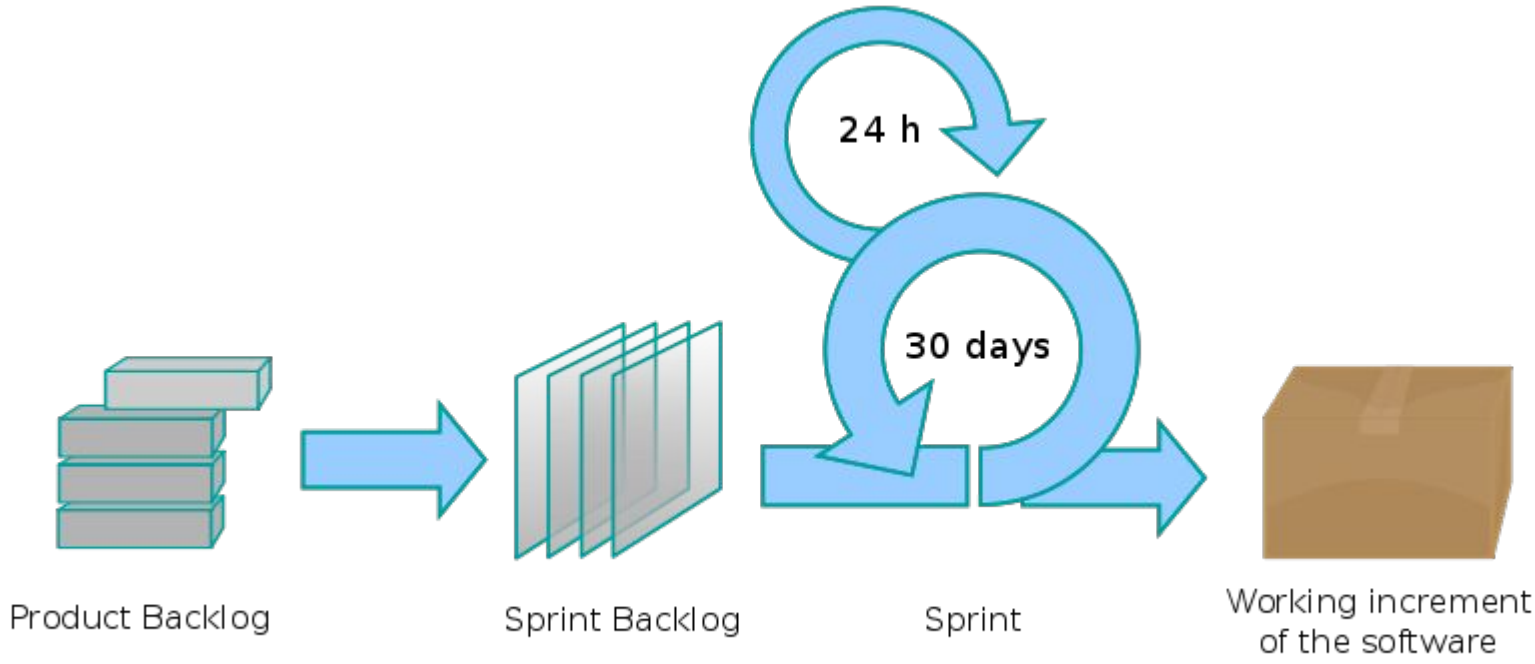
Обзор итогов спринта (Sprint review meeting)

- Проводится после завершения спринта.
- Команда демонстрирует инкремент функциональности продукта всем заинтересованным лицам.
- Привлекается максимальное количество зрителей.
- Все члены команды участвуют в демонстрации (один человек на демонстрацию или каждый показывает, что сделал за спринт).
- Нельзя демонстрировать незавершенную функциональность.
- Ограничена 4-мя часами в зависимости от продолжительности итерации и инкремента

Ретроспективное совещание (Retrospective meeting)

- Проводится после завершения спринта.
- Члены команды высказывают своё мнение о прошедшем спринте.
- Отвечают на два основных вопроса:
  - Что было сделано хорошо в прошедшем спринте?
  - Что надо улучшить в следующем?
- Выполняют улучшение процесса разработки (решают вопросы и фиксируют удачные решения).
- Ограничена 1—3-мя часами.

Sprint (рывок)



Системы управления проектами, поддерживающие scrum

- IBM Rational Team Concert;
- Mingle, ThoughtWorks Studios;
- JIRA (с помощью плагина Green Hopper);
- Redmine and ChiliProject, с помощью плагинов (доступно несколько);
- Visual Studio 2010, Microsoft Team Foundation Server
- Есть и другие инструменты управления проектами, поддерживающие scrum и процессами похожими на scrum.

SCRUM-структура

- В основе Scrum’a лежат ежедневные встречи и 30-дневные циклические встречи, называемые Sprint.
- Все возникающие организационные и технические проблемы всем хорошо видны.
- В конце каждого Sprint’a команда должна демонстрировать заказчику готовый запускаемый файл.
- Требования на функциональность должны были быть определены в начале этого Sprint’a месяц назад и не должны были меняться в течение этого времени.
- Эти требования отображаются в бэклоге — отдельной таблице с указанием того, что нужно сделать и сколько этой займет времени.

Критика

- В agile-подходе часто пренебрегают созданием плана развития продукта, и управлением требованиями, в процессе которого и формируется такой план.
- по сути, управления требованиями просто не существует в данной методологии, а подразумевается возможность заказчика вдруг и неожиданно в конце каждой итерации выставлять новые требования, часто противоречащие архитектуре уже созданного и поставляемого продукта.
- Это иногда приводит к катастрофическим «авралам» с массовым рефакторингом и переделками практически на каждой очередной

- Кроме того, считается, что работа в agile мотивирует разработчиков
  - решать все поступившие задачи простейшим и быстрейшим возможным способом,
  - зачастую не обращается внимания на правильность кода с точки зрения требований нижележащей платформы (подход — «работает, и ладно»)
  - не учитывается, что решение может перестать работать при малейшем изменении или же дать тяжёлые к воспроизводству дефекты после реального развёртывания у клиента.
- Это приводит к снижению качества продукта и накоплению дефектов.

Выводы по гибким процессам разработки ПО

Достоинства	**Недостатки**	**Подходящие типы проектов**
▫ Гибкий и легко реагирующий на изменения. ▫ Короткие циклы предоставления версий ПО. ▫ Непрерывная обратная связь способствовать упрощению внедрения.	▫ Склонность каждый раз к созданию специальных подхода. ▫ М.б проблема в связи с недостатком документации. ▫ Непрерывное изменение кода м.б. рискованным.	Проекты, в которых ▫ часто возникают изменения требований; ▫ потребитель активно участвует в разработке. ▫ размер проекта не слишком большой.