Лекция 12
БГТУ, ФИТ, 1 семестр

Основы программной инженерии (ПИ)

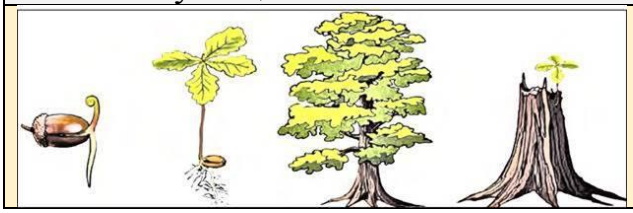
Методологии разработки программного обеспечения

План лекции:
- обзор методологий разработки программного обеспечения.

На прошлой лекции:

1. Жизненный цикл разработки программного обеспечения

Определение:

Жизненный цикл разработки программного обеспечения – это период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент полного его изъятия из эксплуатации.
 это ряд событий, происходящих с ПО в процессе его создания и использования

Модель жизненного цикла программного обеспечения (`Software Development Life Cycle, SDLC`) – это концептуальная структура, которая описывает различные этапы разработки, сопровождения и модификации программного обеспечения.

Основной целью модели жизненного цикла ПО является организация работы над проектом, чтобы убедиться, что каждый этап разработки выполняется в правильной последовательности и с достаточным уровнем детализации.

2. МОДЕЛЬ ЖИЗНЕННОГО ЦИКЛА ПО

1. **подготовка процесса разработки;**
2. **анализ требований к системе;**
3. **проектирование системной архитектуры;**
4. **анализ требований к программным средствам;**
5. **проектирование программной архитектуры;**
6. **техническое проектирование программных средств;**
7. **программирование и тестирование программных средств;**
8. **сборка программных средств;**
9. **тестирование программных средств;**
10. **сборка системы;**
11. **тестрование системы;**
12. **внедрение;**
13. **использование и техническая поддержка.**

Модели разработки ПО:	***модели отличаются по:***
– каскадные; – итерационные; – поэтапные; – другие.	– этапности (фазы, стадии, этапы); – последовательности прохождения этапов (линейная или цикличная); – гибкости (возможность подстраивать процесс под конкретные условия); – связи с определенными методологиями разработки ПО; – использованию инструментальных средств; – другое.

Этапы разработки ПС

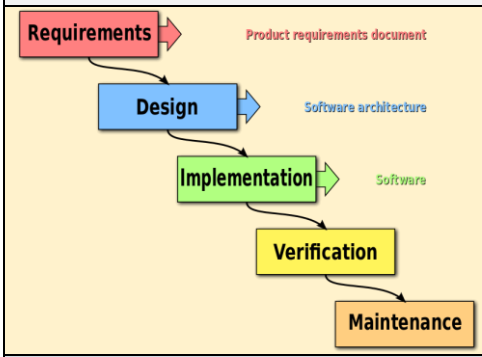
Виды работ	Анализ	Проекти-рование	Кодирование	Отладка и тестирование	Всего
Анализ требований и разработка спецификаций	**13**				**13**
Подготовка данных для отладки		**2**	**2**	**4**	**8**
Планирование отладки	**2**		**2**	**4**	**8**
Проектирование		**13**			**13**
Тестирование	**5**	**5**	**4**	**11**	**25**
Кодирование			**8**		**8**
Испытание ПО				**17**	**17**
Документирование			**4**	**4**	**8**
Всего	**20**	**20**	**20**	**40**	**100**

3. Методологии разработки программного обеспечения

Определение:

Методология разработки ПО –

это система, определяющая порядок выполнения задач, методы оценки и контроля разработки программного обеспечения.



Выбор модели зависит от:

- типа проекта;
- бюджета;
- сроков реализации;
- квалификации команды.

Методологии разработки ПО:

1) Waterfall
2) Rational Unified Process (RUP)
3) Agile – общая методология гибкой разработки
4) Spiral
5) Extreme Programming (XP)
6) Structured Analysis and Design Technique (SADT)
7) Microsoft Solutions Framework (MSF) Microsoft Operations Framework (MOF)
8) Rapid Application Development (RAD)
9) Personal Software Process
10) Scrum – концепция работы в условиях сорванных сроков и идеологического кризиса.

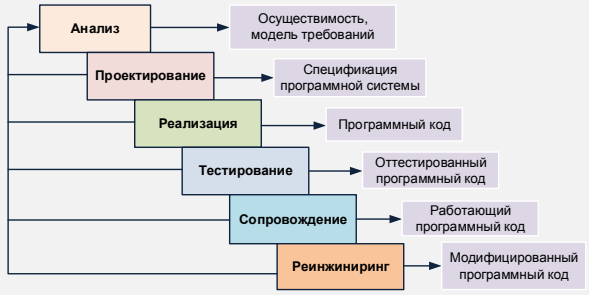
Модель жизненного цикла ПО описывает последовательность этапов и процессов, которые проходит программное обеспечение с момента принятия решения о необходимости создания ПО и заканчивается в момент полного его изъятия из эксплуатации.

Методология разработки ПО представляет собой набор принципов, подходов и практик, которые используются для организации и управления процессом разработки ПО. Методология разработки ПО включет в себя специфичные подходы, такие как Agile, Scrum, Waterfall и другие. Она определяет, как команда разработки организует работу, взаимодействие и управление проектом, а также какие методы и инструменты используются для достижения конечной цели. Методологии разработки ПО могут быть гибкими и адаптивными, позволяя командам быстро реагировать на изменения требований и обратную связь.

4. Методологии разработки ПО

a. Методология, основанная на классической каскадной модели:

Каскадная (Waterfall) модель



- процесс является жестким и линейным;
- порядок этапов строго фиксирован;
- переход на следующий этап происходит после полного завершения работ на предыдущем этапе.
- каждый этап имеет четкие цели.

преимущества:

- оценка качества выполняется после каждого этапа;
- полная и согласованная документация на каждом этапе;
- простота использования;
- стабильные требования;
- бюджет и дедлайны заранее определены.

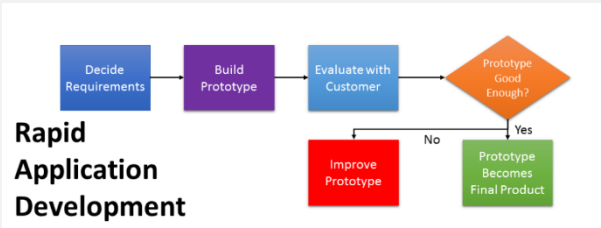
недостатки:

- не очень гибкая система;
- большое количество документации;
- тестирование начинается на последних стадиях;
- результаты работ доступны заказчику только по завершении проекта.

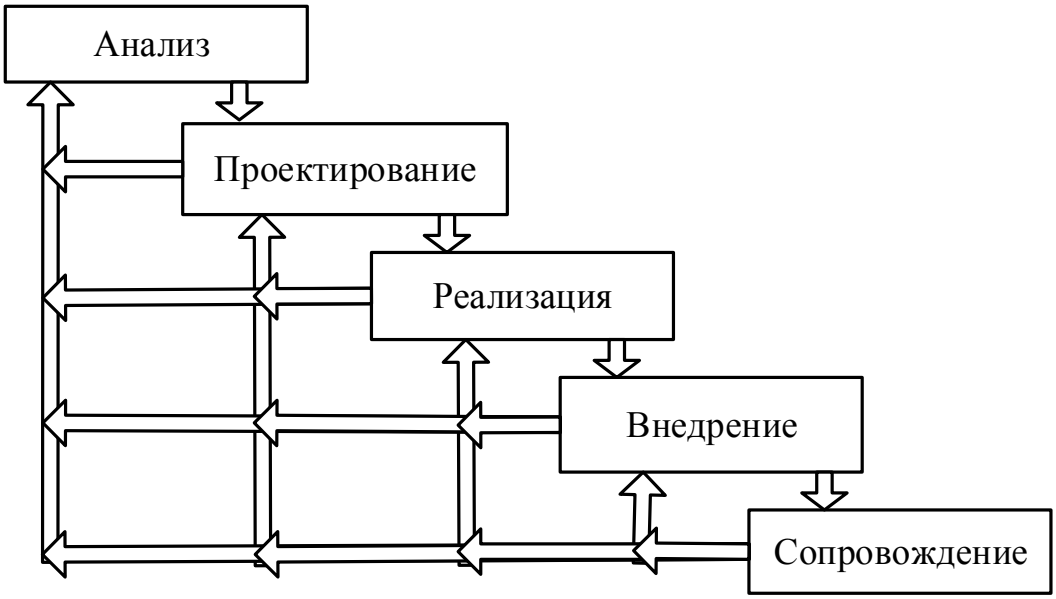
область применения:

- в критически важных системах реального времени;
- в масштабных проектах;
- при разработке новой версии уже существующего продукта или переносе его на новую платформу;
- в софтверных компаниях.

b. *методология быстрой разработки приложений – Rapid Application Development (RAD), основанная на инкрементальной модели*

<table>
<tr>
<td>Rapid Application Development (RAD)
 
</td>
<td>– использование фокус-групп для сбора требований;
– прототипирование и пользовательское тестирование
– повторное использование программных компонентов;
– использование плана, не включающего переработку, или дизайн следующей версии продукта;
– проведение неформальных совещаний по запросу одной из сторон.</td>
</tr>
<tr>
<td>преимущества:
– разработка выполняется быстро и дешево;
– обеспечивается приемлемый для пользователя уровень качества;
– пользователь может оперативно внести изменения в проект;
– функциональность, которая нужна заказчику «еще вчера», можно разработать в первую очередь, и использовать, даже если остальные части программы еще не готовы.</td>
<td>недостатки:
– RAD применима для небольших команд разработчиков;
– RAD зависит от степени участия заказчика в работе проекта.</td>
</tr>
<tr>
<td colspan="2">область применения:
– для проектов, которые легко разделить на независимые или слабосвязанные модули;
– если требования к программному обеспечению быстро меняются;
– в условиях ограниченного бюджета;
– нет ясного представления, как должен выглядеть и работать продукт;
– разработка ведется командой профессионалов;
– если пользователь готов активно участвовать в проекте на протяжении всей работы.</td>
</tr>
</table>

Жизненный цикл ПО по методологии RAD состоит из фаз:



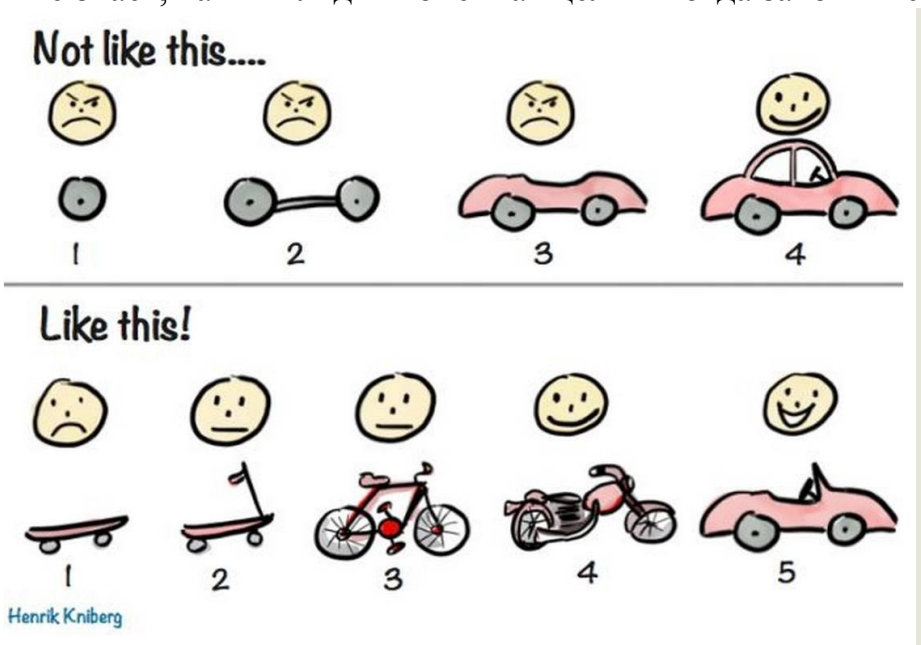
c. инкрементная модель

Incremental Model  	Метод, в котором ПО проектируется, реализуется и тестируется инкрементно (каждый раз с небольшими добавлениями) до самого окончания разработки;
преимущества: – быстрый выпуск минимального продукта; – ошибка обходится дешевле; – постоянное тестирование пользователями.	***недостатки:*** – требуется переработка проекта; – отсутствие фиксированного бюджета и сроков.
область применения: – когда основные требования к системе четко определены и понятны, некоторые детали могут дорабатываться с течением времени (поэтапно); – требуется ранний вывод продукта на рынок; – при разработке веб-приложений и продуктов компаний-брендов.	

d. Итеративная или итерационная модель (Iterative Model)

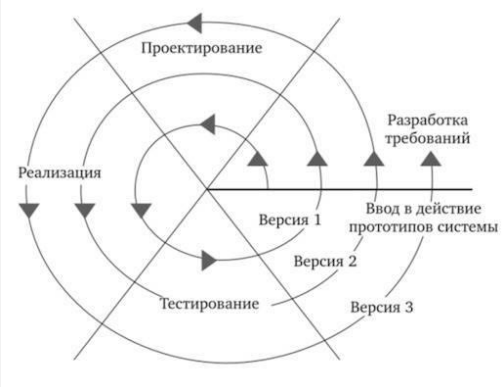
<table>
<tr>
<td>Iterative Model
 
</td>
<td>– определение и анализ требований;
– дизайн и проектирование: согласно требованиям. Причем дизайн может как разрабатываться отдельно для данной функциональности, так и дополнять уже существующий;
– разработка и тестирование: кодирование, интеграция и тестирование нового компонента;
– фаза ревью: оценка, пересмотр текущих требований.</td>
</tr>
<tr>
<td>преимущества:
– быстрый выпуск минимального продукта позволяет оперативно получать обратную связь от заказчика и пользователей;
– постоянное тестирование пользователями позволяет быстро обнаруживать и устранять ошибки.</td>
<td>недостатки:
– переработка проекта;
– заказчик не знает, как выглядит конечная цель и когда закончится разработка;
– отсутствие фиксированного бюджета и сроков.</td>
</tr>
<tr>
<td colspan="2">область применения:
– требования к конечной системе заранее четко определены и понятны;
– для работы над большими проектами с неопределёнными требованиями;
– для задач с инновационным подходом, когда заказчик не уверен в результате.</td>
</tr>
</table>

Заказчик не знает, как выглядит конечная цель и когда закончится разработка:

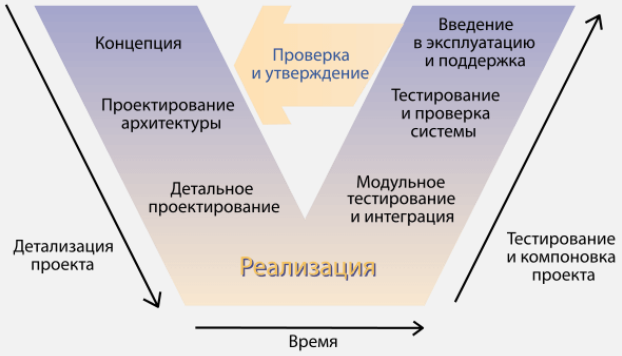


Отличие итерационной модели от инкрементной:

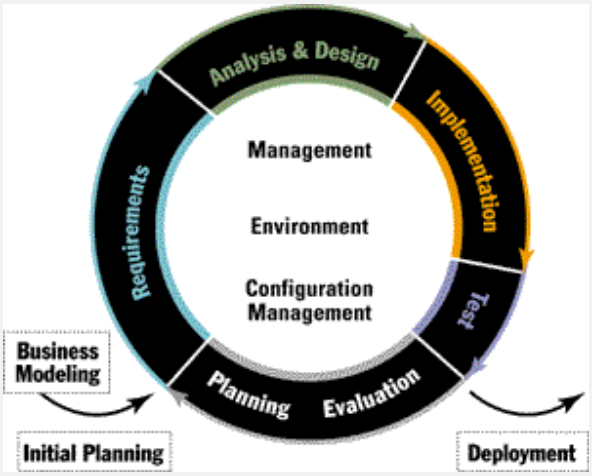
e. Спиральная модель (Spiral Model)

<table>
<tr>
<td>Spiral Model
 
</td>
<td>– заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей;
– в конце каждого витка (цикла разработки) итераций принимается решение, продолжать ли проект.</td>
</tr>
<tr>
<td>преимущества:
– большое внимание уделяется проработке рисков.</td>
<td>недостатки:
– есть риск застрять на начальном этапе: бесконечно совершенствовать первую версию продукта и не продвинуться к следующим;
– разработка длится долго и стоит дорого.</td>
</tr>
<tr>
<td colspan="2">область применения:
– для решения критически важных бизнес-задач.</td>
</tr>
</table>

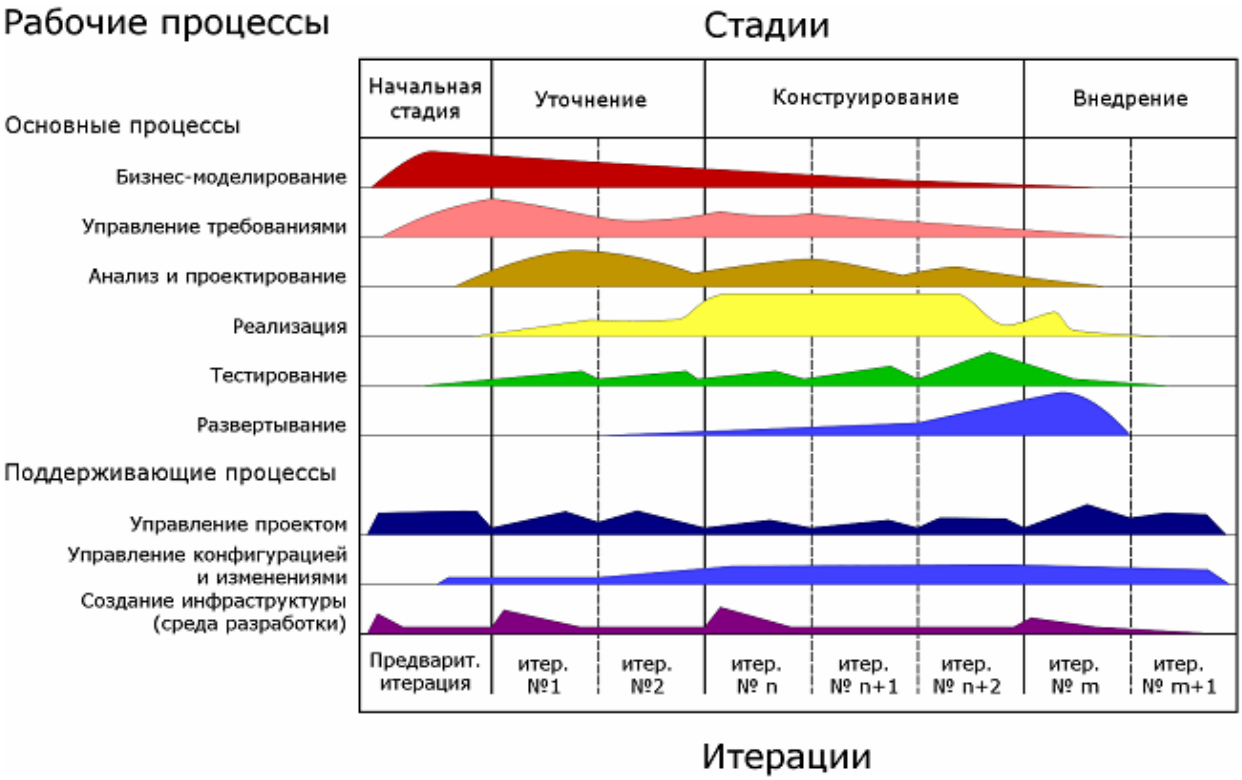
f. V-образная модель (разработка через тестирование)

<table>
<tr>
<td>V-Model (validation and verification)
 
</td>
<td>заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе.
Это усовершенствованная каскадная модель</td>
</tr>
<tr>
<td>преимущества:
– количество ошибок в архитектуре ПО сводится к минимуму.</td>
<td>недостатки:
– если была допущена ошибка при разработке архитектуры, то вернуться и исправить её будет стоить дорого, как и в Waterfall.</td>
</tr>
<tr>
<td colspan="2">область применения:
– для малых и средних проектов, где требования четко определены и фиксированы;
– если требуется тщательное тестирование;
– для проектов, в которых важна надёжность и цена ошибки очень высока.</td>
</tr>
</table>

g. методология «Рациональный унифицированный процесс (Rational Unified Process (RUP)»

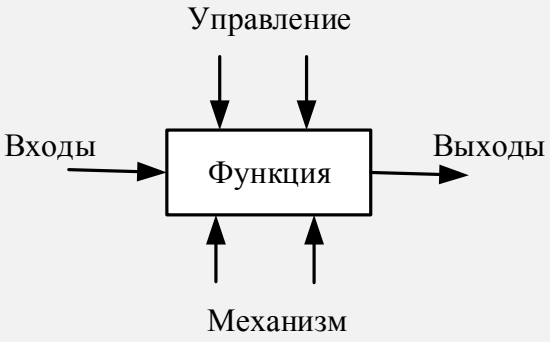
<table>
<tr>
<td>Rational Unified Process (RUP)
 
</td>
<td>RUP основные принципы:
– итеративная модель разработки;
– управление требованиями;
– компонентная архитектура;
– визуальное моделирование ПО;
– проверка качества ПО;
– контроль внесенных изменений.</td>
</tr>
<tr>
<td>преимущества:
– изменения и уточнения требований выявляются в ранний период разработки;
– заказчик участвует в оценке промежуточных версий (прототипов) ПО;
– снижены архитектурные и интеграционные риски;
– повторное использование программных элементов;
– точная документация.</td>
<td>недостатки:
– высокие расходы;
– сложность внедрения.</td>
</tr>
<tr>
<td colspan="2">область применения:
– применима в небольших и быстрых проектах, где за счет отсутствия формализации требуется сократить время выполнения проекта и расходы;
– применима в больших и сложных проектах, где требуется высокий уровень формализма.</td>
</tr>
</table>

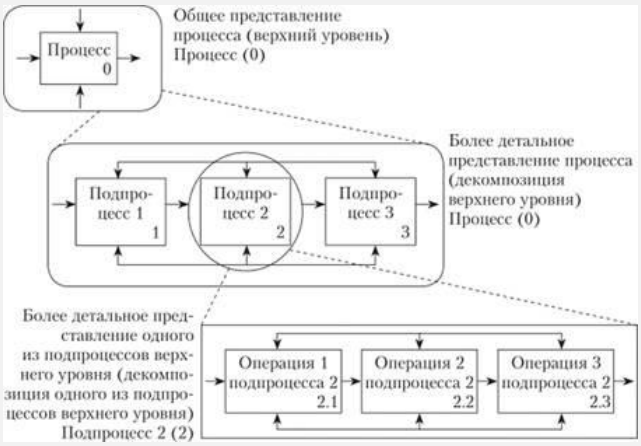
Полный жизненный цикл разработки продукта состоит из четырёх фаз, каждая из которых включает в себя одну или несколько итераций:



h. методология «Structured Analysis and Design Technique (SADT)»

Structured Analysis and Design Technique (SADT)





Методология SADT:

разработана специально для описания искусственных систем средней сложности.

На методологии SADT основана группа методологий IDEF:

- IDEF0 – для создания функциональной модели, отражающей структуру и функции системы, а также потоки информации и материальных объектов, преобразуемые этими функциями;
- IDEF1 – для построения информационной модели, отражающей структуру и содержание информационных потоков, необходимых для поддержки функций системы;
- IDEF2 позволяет построить динамическую модель изменяющихся во времени функций, информации и ресурсов системы.

преимущества:

- универсальность;
- отражает такие характеристики, как управление, обратная связь и исполнители;
- процедуры поддержки коллективной работы;
- может быть использована на ранних этапах создания системы (предпроектная стадия);
- сочетается с другими структурными методами проектирования.

недостатки:

- сложность восприятия диаграмм;
- большое количество уровней декомпозиции;
- трудность увязки нескольких процессов.

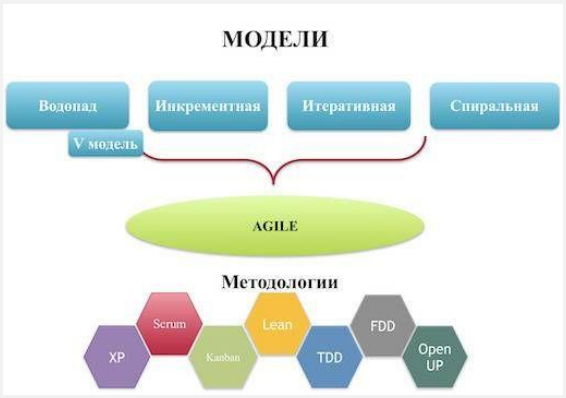
область применения:

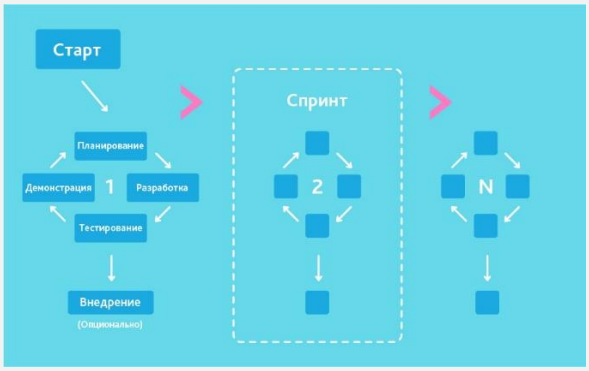
- используется при описании банковского дела, организации материально-технического снабжения, построении системы планирования производства.

i. методология гибкой разработки ПО (Agile Model)

<table>
<tr>
<td>

Agile Model





</td>
<td>

Включает в себя:

1. экстремальное программирование (Extreme Programming, XP);
2. бережливую разработку программного обеспечения (Lean);
3. фреймворк для управления проектами Scrum;
4. разработку, управляемую функциональностью (Feature-driven development, FDD);
5. разработку через тестирование (Test-driven development, TDD);
6. методологию «чистой комнаты» (Cleanroom Software Engineering);
7. итеративно-инкрементальный метод разработки (OpenUP);
8. методологию разработки Microsoft Solutions Framework (MSF);
9. метод разработки динамических систем (Dynamic Systems Development Method, DSDM);
10. метод управления разработкой Kanban.

</td>
</tr>
<tr>
<td>

преимущества:

– гибкость;
– позволяет получить более совершенный продукт, поскольку проверка качества осуществляется по окончании каждого спринта;
– быстрый выпуск продукта.

</td>
<td>

недостатки:

– сроки сдачи готового проекта могут постоянно переноситься;
– необходимость включения соответствующих корректировок в проектную документацию;
– ежедневные встречи членов рабочей команды;
– отсутствие стабильных требований к конечному результату.

</td>
</tr>
<tr>
<td colspan="2">

область применения:

– для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка.

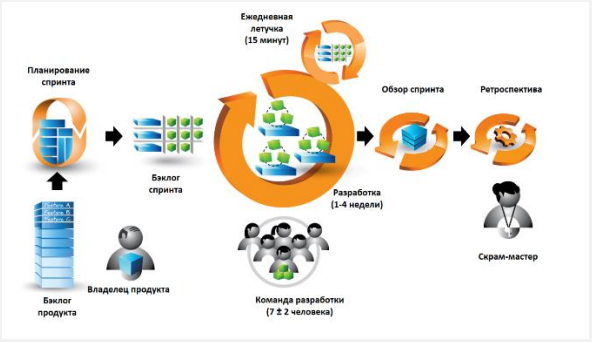
</td>
</tr>
</table>

Различия между Agile и традиционным подходом к разработке:

Характеристики проекта	Традиционные модели	Гибкие методологии
Подход	Прогнозирующий	Адаптивный
Критерии успеха	Следование плану	Ценность для бизнеса
Риски	Риски определены	Риски не определены
Контроль	Легко контролировать	Зависит от профессионального уровня специалистов
Заказчики	Низкая вовлеченность	Высокая вовлеченность
Документация	Детальная с начала проекта	Доработка по мере развития проекта
Требования	Известны заранее, стабильны	Не всегда известны заранее, легко изменяемы
Команда проекта	Включение новых специалистов на любом этапе	Опытные специалисты, стабильный состав
Рефакторинг (изменение кода)	Дорого	Недорого

Agile/Scrum

Agile/Scrum



Scrum кратко: agile-подход к разработке и управлению проектами:

- деление работы на части, которые называются спринтами (две недели);
- спринты планируются исходя из требований для данного момента;
- относительная оценка времени выполнения работ;
- ревью каждого спринта, чтобы понять, как он прошёл и что можно было бы улучшить;
- фидбек (обратная связь) по поставляемому продукту;
- ежедневные собрания (15 мин.).

Agile/Scrum:

- ***Scrum-команда*** – это команда, работающая над проектом (разработчики, тестеры, дизайнеры).
- ***Scrum-мастер*** – организатор процесса, который следит, чтобы соблюдались принципы скрама.
- ***Product owner*** – заказчик.

Митинги:

Stand-up (короткий митинг, проводится каждый день; какие задачи должны быть выполнены каждым: Что делал? Что будет делать? Какие есть проблемы?) →
Плэннинг (определяют какие задачи должны быть выполнены за следующий спринт) →
Ретроспектива (проводится в конце спринта)

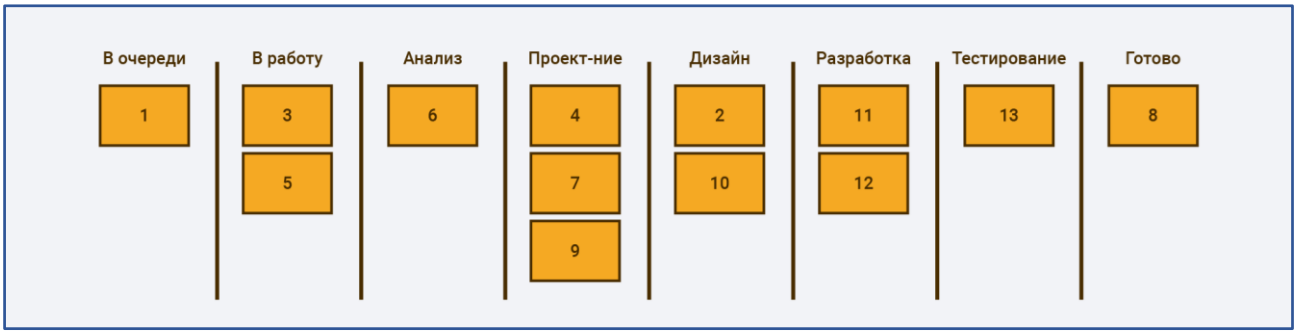
недостатки:

- успех проекта во многом зависит от скрам-мастера, квалификации команды и их приверженности своему делу;
- далеко не всегда можно адаптировать метод скрам под сферу деятельности, поскольку есть проекты, требующие исключительно планового подхода в работе;
- требует регулярной коммуникации с заказчиком, что порой тормозит процесс из-за невозможности получения обратной связи;
- сложность внедрения в масштабных и сложных проектах, так как больше подходит для малых и средних.

область применения:

- строительство, образование, производство товаров, Event-индустрия и другие виды деятельности.

Agile/Kanban



Agile/Kanban (с японского переводится, как «карточка»)

Kanban кратко: agile-подход к разработке и управлению проектами, ориентированный на задачи:

- еженедельные собрания;
- непрерывная разработка;
- визуализация процесса на доске;
- решение сначала самых важных задач;
- поэтапные улучшения.

Цель: анализ рабочего процесса и поиск точек для улучшения.

преимущества:

- простота использования;
- наглядность (помогает в определение узких мест, упрощает понимание);
- высокая вовлеченность команды в сам процесс;
- высокая гибкость в разработке.

недостатки:

- нестабильный список задач;
- сложно применять на долгосрочных проектах;
- отсутствие жестких дедлайнов.

область применения:

- когда потребности пользователей постоянно меняются в динамическом бизнесе;
- отлично себя показывает в сферах неосновного производства.
- хорошо работает при управлении стартапами без четкого плана, но где активно продвигается разработка.

Agile/Extreme Programming (XP)

Extreme Programming (XP)

Экстремальное программирование – возможность вести разработку в условиях постоянно меняющихся требований.

Основные принципы:

- итеративность: разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком;
- простота решений: принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения;
- интенсивная разработка малыми группами (не больше 10 человек) и парное программирование;
- обратная связь с заказчиком;
- достаточная степень смелости и желание идти на риск.

преимущества:

- заказчик получает именно тот продукт, который ему нужен;
- команда быстро вносит изменения в код и добавляет новую функциональность;
- код всегда работает за счет постоянного тестирования;
- высокое качество кода;
- код написан по единому стандарту и постоянно рефакторится;
- быстрый темп разработки за счет парного программирования;
- снижаются риски, связанные с разработкой.

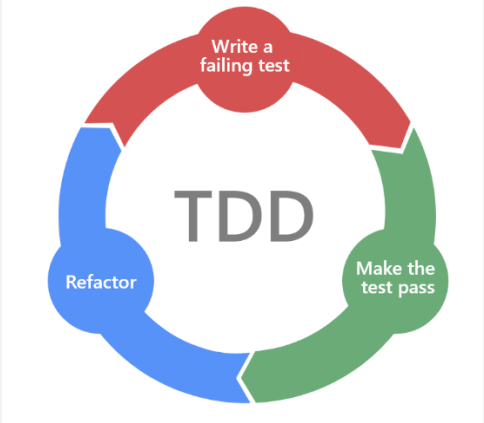
недостатки:

- успех проекта зависит от вовлеченности заказчика;
- сложность планирования;
- успех XP сильно зависит от уровня программистов;
- регулярные встречи с программистами дорого обходятся заказчикам.

область применения:

- из-за недостатка структуры и документации не подходит для крупных проектов.

Разработка через тестирование (Test-driven development, Agile/TDD)

<table>
<tr>
<td>Test-driven development, TDD
 
</td>
<td>Цикл разработки по TDD:
– добавить тест для новой (еще не реализованной) функциональности или для воспроизведения существующего бага;
– запустить все тесты и убедиться, что новый тест не проходит;
– написать код, который обеспечит прохождение теста;
– запустить тесты и убедиться, что они все прошли успешно;
– заняться рефакторингом и оптимизацией;
– перезапустить тесты и убедиться, что они все ещё проходят успешно;
– повторить цикл.</td>
</tr>
<tr>
<td>преимущества:
– самотестирующийся код, это означает, что код работает именно так, как нужно программисту;
– возможность постоянного и быстрого, в режиме реального времени, получения сведений о состоянии кода;
– тесты могут использоваться в качестве документации.</td>
<td>недостатки:
– возрастающая сложность для языков с динамической типизацией.</td>
</tr>
<tr>
<td colspan="2">область применения:
– подходит для всех проектов.</td>
</tr>
</table>

Microsoft Solutions Framework (MSF) и Microsoft Operations Framework (MOF)

Microsoft Solutions Framework (MSF)

Основные принципы:

- способствуйте открытому общению;
- работайте над общим видением;
- расширяйте полномочия членов команды;
- разделяйте ответственность;
- сотрудничайте с клиентом и сосредоточьтесь на предоставлении бизнес-ценности;
- будьте готовы к переменам и оставайтесь гибкими;
- инвестируйте в качество;
- учитесь на опыте.

преимущества:

- гибкость;
- масштабируемость;
- включает основополагающие принципы, модели и дисциплины управления персоналом, процессами и технологиями.

недостатки:

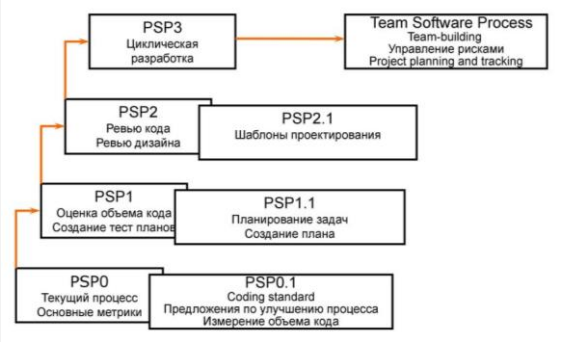
- MSF не описывает детально важнейшие роли заказчика и пользователя;
- не рассматриваются также методы управления группой проектов, из которых состоит основной проект.

область применения:

- подходит для работы в проектной группе или организации любого масштаба.

Модель компетентного разработчика (Personal Software Process)

Personal Software Process



Personal Software Process определяет требования к компетенциям разработчика

Цели PSP

PSP помогает разработчикам:

- улучшить оценку и планирование навыков;
- управлять качеством проектов;
- снизить количество ошибок в своих разработках.

Один из основных аспектов PSP — использование накопленной статистики для анализа и улучшения показателей процесса разработки. Сбор статистики включает 4 элемента:

- скрипты.
- оценки. Включают 4 основных элемента:
  - размер — оценка размера для части продукта. Например, количество строк кода (LOC — Lines Of Code).
  - качество — количество ошибок в продукте.
  - усилия — оценка времени, требующегося для завершения задачи, обычно записываемое в минутах.
  - планирование — оценка хода проекта, перемещаемая между планируемыми и завершенными пунктами.
- стандарты кодирования. Применение стандартов к процессу может обеспечить точные и постоянные данные.
- формы.

Personal Software Process