

Введение в разработку программного обеспечения (ИС и ЦД)

**Жизненный цикл разработки программного обеспечения
Методологии разработки программного обеспечения**

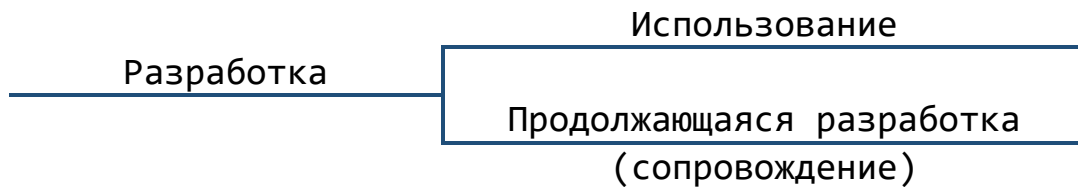
План лекции:

- понятие жизненного цикла разработки ПО;
- основные определения;
- взаимосвязь между процессами жизненного цикла;
- стандарт ISO/IEC 12207: 1995.
- обзор методологий разработки программного обеспечения.

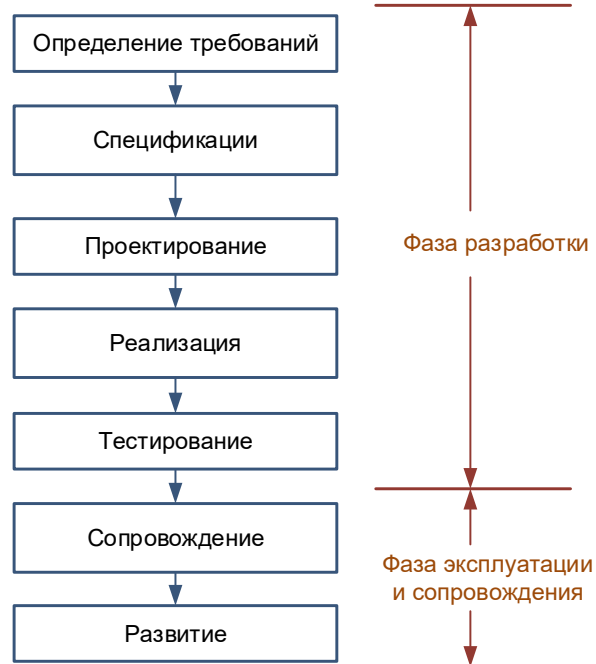
1. Жизненный цикл разработки программного обеспечения

Общепринятая модель жизненного цикла программного обеспечения, согласно которой программные системы проходят в своем развитии два этапа:

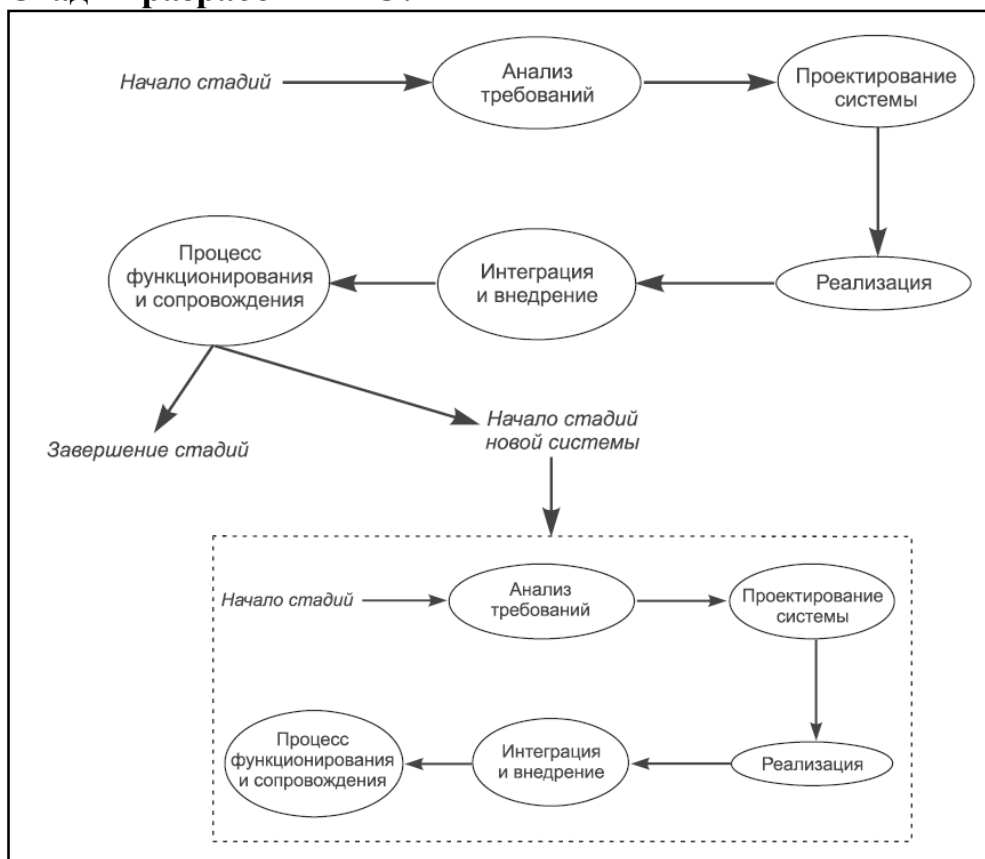
- ✓ разработка;
- ✓ сопровождение.



Каждая стадия разбивается на ряд этапов:



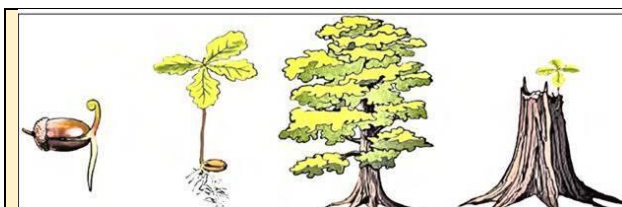
Стадии разработки ПО:



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.

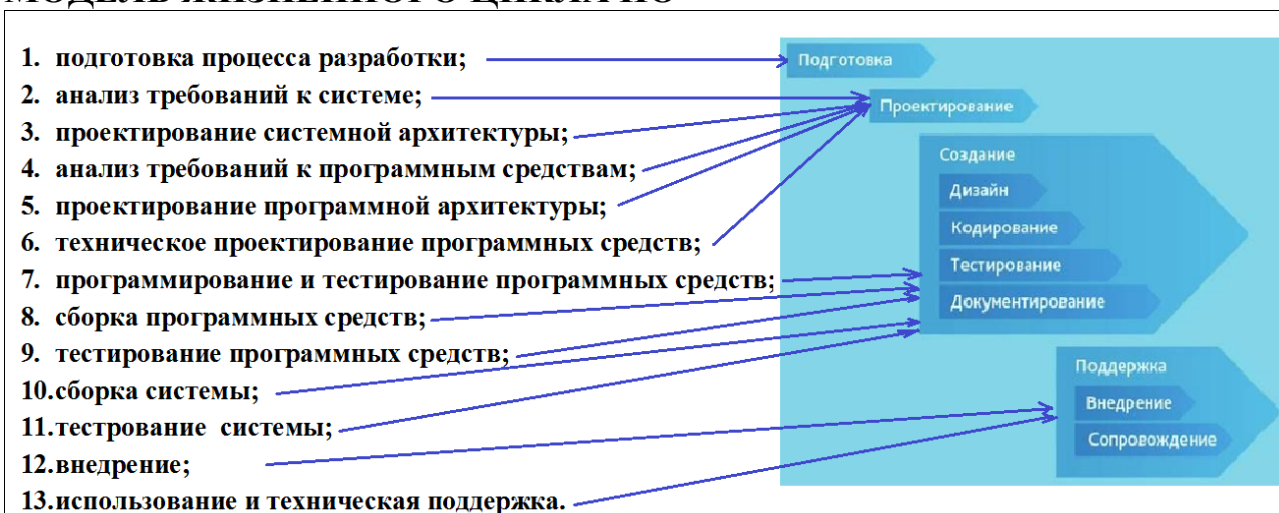


это ряд событий, происходящих с ПО в процессе его создания и использования

Назначение

<p>Назначение модели жизненного цикла ПО</p>	<ul style="list-style-type: none"> ✓ <i>дает рекомендации по организации процесса разработки ПО в целом, конкретизируя его до видов деятельности, артефактов, ролей и их взаимосвязей</i> ✓ <i>служит основой для планирования программного проекта</i> ✓ <i>способствует правильному распределению обязанностей сотрудников</i>
---	---

МОДЕЛЬ ЖИЗНЕННОГО ЦИКЛА ПО



<p>Модели разработки ПО:</p> <ul style="list-style-type: none"> – каскадные; – итерационные; – поэтапные; – другие. 	<p><i>модели отличаются по:</i></p> <ul style="list-style-type: none"> – этапности (фазы, стадии, этапы); – последовательности прохождения этапов (линейная или цикличная); – гибкости (возможность подстраивать процесс под конкретные условия); – связи с определенными методологиями разработки ПО; – использованию специализированных инструментальных средств; – другие.
--	---

Стандарт ISO/IEC 12207:1995 – Information Technology – Software Life Cycle Process

- принят в 1995 году ISO (International Organization for Standardization) совместно с IEC (International Electrotechnical Commission – Международная электротехническая комиссия)
- в 1999г. он был принят как ГОСТ (ИСО/МЭК) 12207 – Процессы жизненного цикла программных средств (последнее издание 2010г.)

определяет организацию ЖЦ программного продукта как совокупность процессов, каждый из которых разбит на действия, состоящие из отдельных задач; устанавливает структуру (архитектуру) ЖЦ программного продукта в виде перечня процессов, действий и задач.

Определения (стандарт ISO/IEC 12207):

Процесс	совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные
Каждый процесс разделен на набор действий; Каждое действие – на набор задач.	
Каждый процесс характеризуется:	<ul style="list-style-type: none">✓ задачами и методами их решения;✓ исходными данными;✓ результатами.

Структура процессов жизненного цикла программного обеспечения:


Основные процессы	<ul style="list-style-type: none">✓ приобретение✓ поставка✓ разработка✓ эксплуатация✓ сопровождение
Организационные процессы	<ul style="list-style-type: none">✓ управление✓ усовершенствование✓ создание инфраструктуры✓ обучение
Вспомогательные процессы	<ul style="list-style-type: none">✓ документирование✓ управление конфигурацией✓ обеспечение качества✓ верификация✓ аттестация✓ совместная оценка✓ аудит✓ разрешение проблем

Процесс разработки включает следующие действия:

- подготовительную работу;
- анализ требований к системе;
- проектирование архитектуры системы;
- анализ требований к ПО;
- проектирование архитектуры ПО;
- детальное проектирование ПО;
- кодирование и тестирование ПО;
- интеграцию ПО;
- квалификационное тестирование ПО;
- интеграцию системы;
- квалификационное тестирование системы;
- установку ПО;
- приемку ПО.

2. Виды моделей жизненного цикла

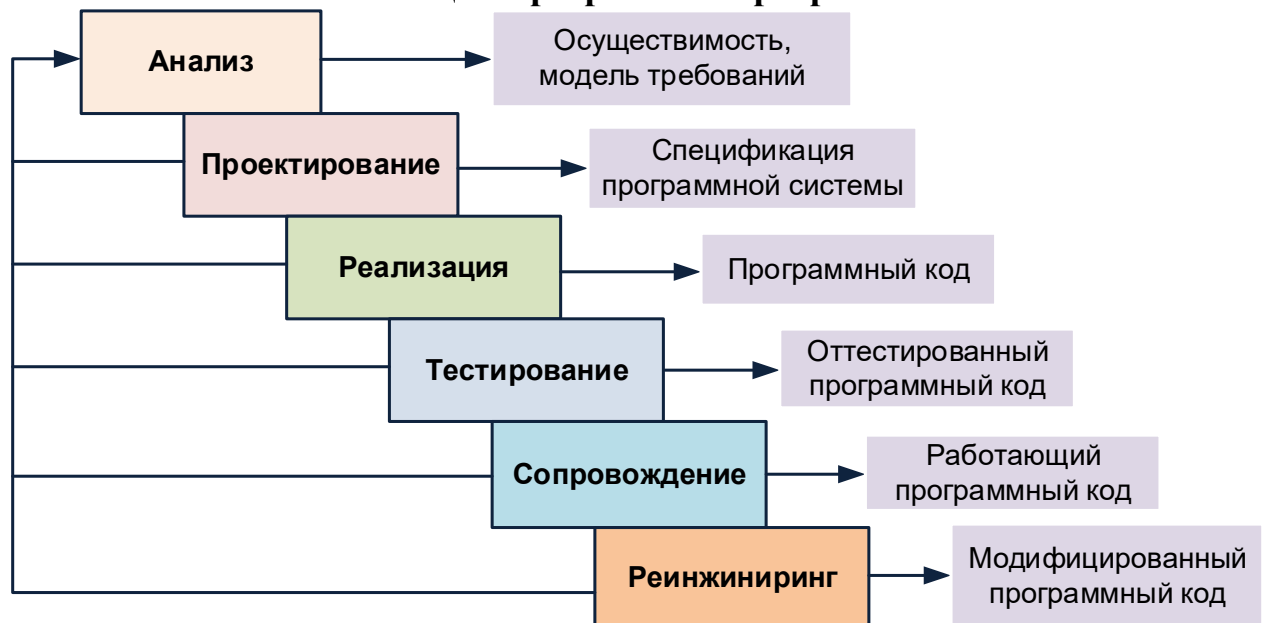
В настоящее время используются следующие модели жизненного цикла:

<p>Каскадная (водопадная) модель</p> 	<ul style="list-style-type: none">– последовательное и однократное выполнение всех этапов проекта в строго фиксированном порядке;– переход на следующий этап после полного завершения работ на предыдущем этапе.
<p>преимущества:</p> <ul style="list-style-type: none">– на каждой стадии формируется законченный набор проектной документации;– выполняемые в логической последовательности стадии работ позволяют планировать сроки завершения всех работ и соответствующие затраты.	<p>недостатки:</p> <ul style="list-style-type: none">– выявление и устранение ошибок производится только на стадии тестирования (как следствие, неточные спецификации приводят к переработке уже принятых решений);– реальные проекты часто требуют отклонения от стандартной последовательности шагов;– ЖЦ основан на точной формулировке исходных требований к ПО (на практике часто случается, что в начале проекта требования заказчика определены лишь частично);– результаты работ доступны заказчику только по завершении проекта.

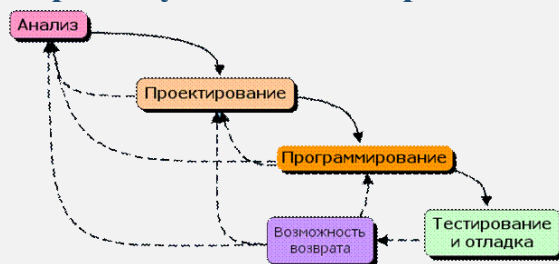
Область применения каскадной модели:

- в критически важных системах реального времени (например, управление авиационным движением или медицинским оборудованием);
- в масштабных проектах, в реализации которых задействовано несколько больших команд разработчиков;
- при разработке новой версии уже существующего продукта или переносе его на новую платформу;
- в организациях, имеющих большой практический опыт в создании программных систем определенного типа (например, бухгалтерский учет, начисление зарплаты и пр.).

Классический жизненный цикл разработки программного обеспечения:



Поэтапная модель с промежуточным контролем



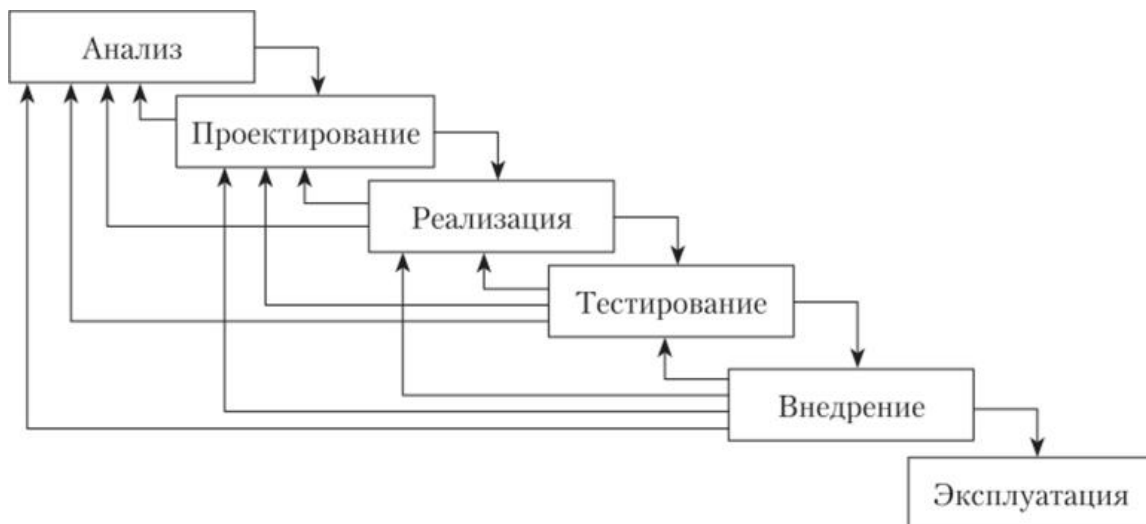
- разработка ведется итерациями с циклами обратной связи между этапами;
- корректировки между этапами учитывают существующее взаимосвязи результатов разработки на различных этапах;
- время жизни каждого из этапов растягивается на весь период разработки.

преимущества:

- оперативная разработка и демонстрация ПО заказчику для устранения ошибок;
- допускается отсутствие требований к ПО.

недостатки:

- сложность планирования работ и сроков завершения проекта;
- ориентирование на разработку.



Спиральная модель



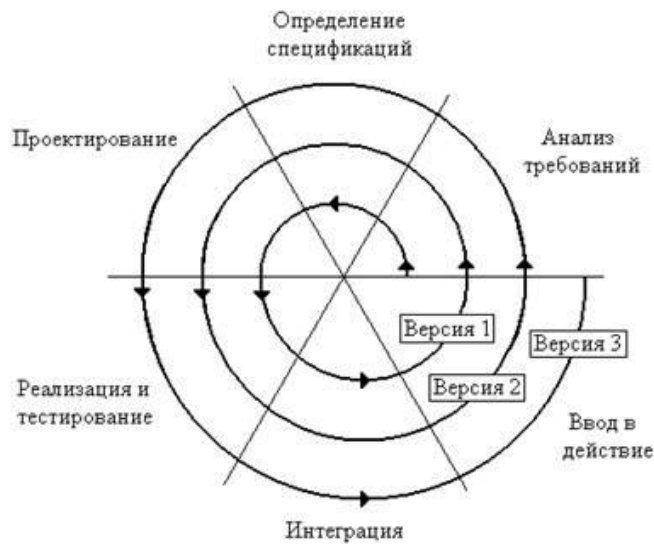
- на каждом витке спирали создается очередная версия продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка.

преимущества:

- прототипирование позволяет пользователям «увидеть» систему на ранних этапах разработки;
- позволяет идентифицировать риски без особых дополнительных затрат;
- предусматривает активное участие пользователей в работах по планированию, анализу рисков, разработке и оценке полученных результатов.

недостатки:

- модель имеет усложненную структуру;
- сложность поддержания версий продукта;
- сложность оценки точки перехода на следующий цикл;
- спираль может продолжаться до бесконечности, поскольку каждая ответная реакция заказчика на созданную версию может порождать новый цикл, что отдалает окончание работы над проектом.



Область применения спиральной модели

- ✓ при разработке систем, требующих большого объема вычислений (например, систем принятия решений);
- ✓ при выполнении бизнес-проектов;
- ✓ при выполнении проектов в области аэрокосмической промышленности, обороны и инжиниринга, где уже имеется позитивный опыт ее использования.

Другие модели:

<p>каркасная модель разработки</p>	
<p>сборочное программирование</p>	
<p>исследовательское программирование</p>	

Разные типы проектов требуют разных сочетаний подготовки и конструирования ПО. Каждый проект уникален, однако обычно проекты подпадают под общие стили разработки.

Можно выделить три самых популярных типа проектов и в большинстве случаев оптимальные методы работы над ними.

	Бизнес-системы	Системы целевого назначения	Встроенные системы, от которых зависит жизнь людей
Типичные приложения	Интернет-сайты. Сайты в интрасетях. Игры. Системы управления информацией. Системы управления информацией. Системы выплаты заработной платы. ...	Встроенное ПО. Игры. Интернет-сайты. Встроенное ПО. Web-сервисы. ...	Авиационное ПО. Встроенное ПО. ПО для медицинских устройств. Операционные системы. Пакетное ПО. ...
Модели жизненного цикла	Гибкая разработка (экстремальное программирование, методология Scrum, Эволюционное прототипирование).	Поэтапная модель. Эволюционная модель. Спиральная разработка.	Поэтапная модель. Спиральная разработка. Эволюционная модель.
Внедрение приложения	Неформальная процедура внедрения	Формальная процедура внедрения.	Формальная процедура внедрения.

Этапы разработки ПС

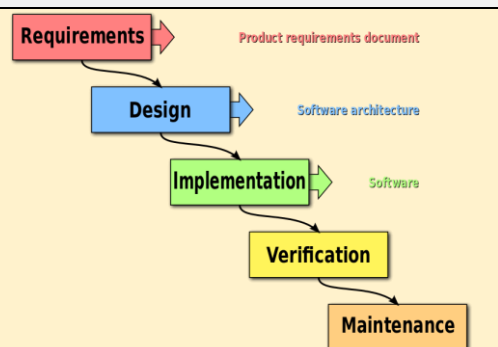
Виды работ	Анализ	Проектирование	Кодирование	Отладка и тестирование	Всего
Анализ требований и разработка спецификаций	13				13
Подготовка данных для отладки		2	2	4	8
Планирование отладки	2		2	4	8
Проектирование		13			13
Тестирование	5	5	4	11	25
Кодирование			8		8
Испытание ПО				17	17

Документирование			4	4	8
Всего	20	20	20	40	100

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.

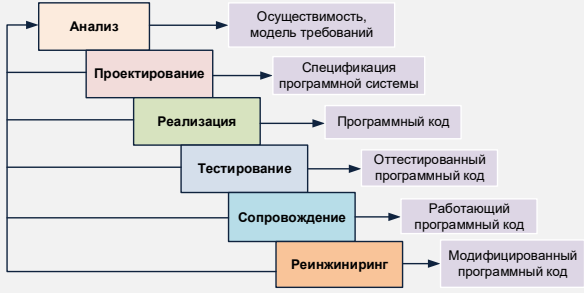
Методологии разработки ПО:

- 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.

3. Методологии разработки ПО

а. Методология, основанная на классической каскадной модели:

Каскадная (Waterfall) модель	<ul style="list-style-type: none"> – процесс является жестким и линейным; – порядок этапов строго фиксирован; – переход на следующий этап происходит после полного завершения работ на предыдущем этапе.
-------------------------------------	---

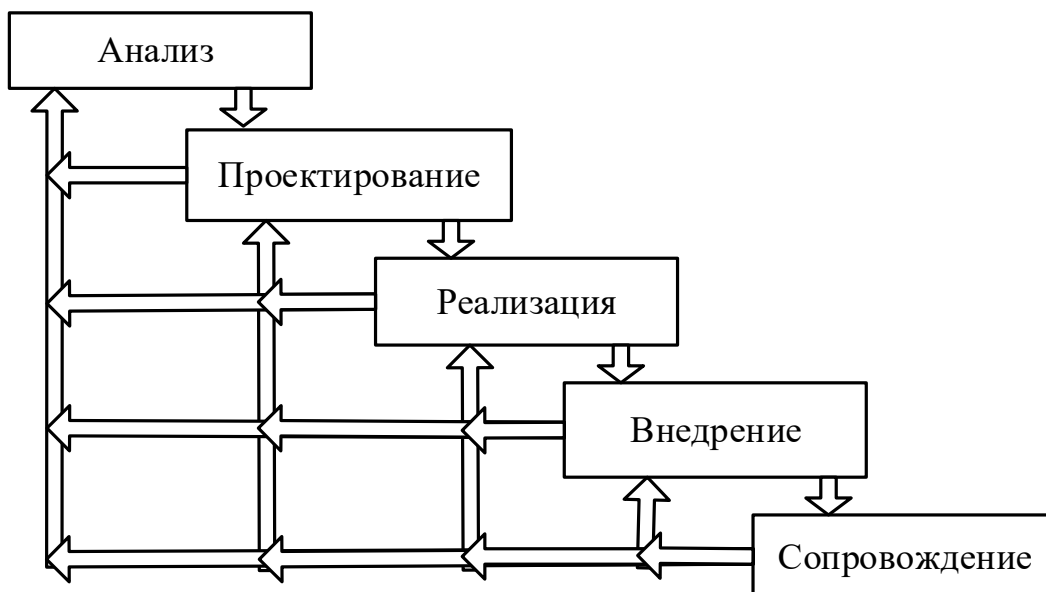
	<ul style="list-style-type: none"> — каждый этап имеет четкие цели.
<p>преимущества:</p> <ul style="list-style-type: none"> — оценка качества выполняется после каждого этапа; — полная и согласованная документация на каждом этапе; — простота использования; — стабильные требования; — бюджет и дедлайны заранее определены. 	<p>недостатки:</p> <ul style="list-style-type: none"> — не очень гибкая система; — большое количество документации; — тестирование начинается на последних стадиях; — результаты работ доступны заказчику только по завершении проекта.
<p>область применения:</p> <ul style="list-style-type: none"> — 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


<p>Rapid Application Development (RAD)</p> 	<ul style="list-style-type: none"> — 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.
---	--

<p>преимущества:</p> <ul style="list-style-type: none"> – 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. 	<p>недостатки:</p> <ul style="list-style-type: none"> – RAD применима для небольших команд разработчиков; – RAD зависит от степени участия заказчика в работе проекта.
<p>область применения:</p> <ul style="list-style-type: none"> – 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. 	

Жизненный цикл ПО по методологии RAD состоит из фаз:



с. инкрементная модель

<p>Incremental Model</p> 	<p>Метод, в котором ПО проектируется, реализуется и тестируется инкрементно (каждый раз с небольшими добавлениями) до самого окончания разработки;</p>
<p>преимущества:</p> <ul style="list-style-type: none"> – быстрый выпуск минимального продукта; – ошибка обходится дешевле; – постоянное тестирование пользователями. 	<p>недостатки:</p> <ul style="list-style-type: none"> – требуется переработка проекта; – отсутствие фиксированного бюджета и сроков.
<p>область применения:</p> <ul style="list-style-type: none"> – 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)

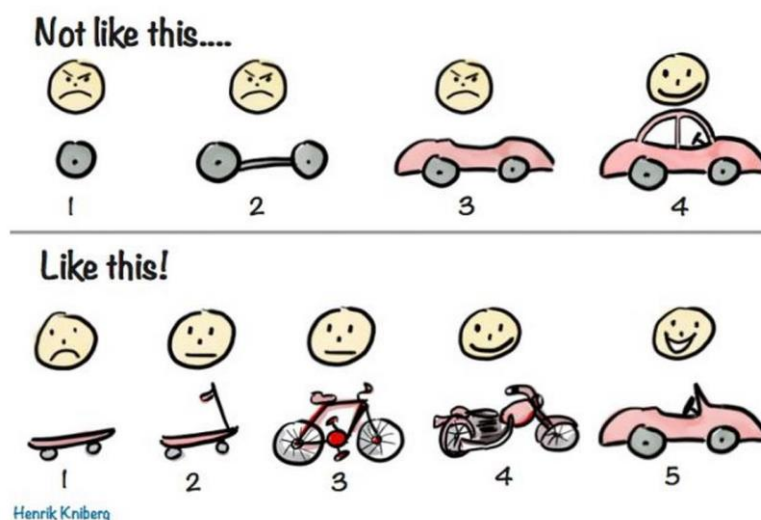
<p>Iterative Model</p> 	<ul style="list-style-type: none"> – определение и анализ требований; – дизайн и проектирование: согласно требованиям. Причем дизайн может как разрабатываться отдельно для данной функциональности, так и дополнять уже существующий; – разработка и тестирование: кодирование, интеграция и тестирование нового компонента; – фаза ревью: оценка, пересмотр текущих требований.
<p>преимущества:</p> <ul style="list-style-type: none"> – быстрый выпуск минимального продукта позволяет оперативно 	<p>недостатки:</p> <ul style="list-style-type: none"> – переработка проекта;

<p>получать обратную связь от заказчика и пользователей;</p> <p>– постоянное тестирование пользователями позволяет быстро обнаруживать и устранять ошибки.</p>	<p>– заказчик не знает, как выглядит конечная цель и когда закончится разработка;</p> <p>– отсутствие фиксированного бюджета и сроков.</p>
--	--

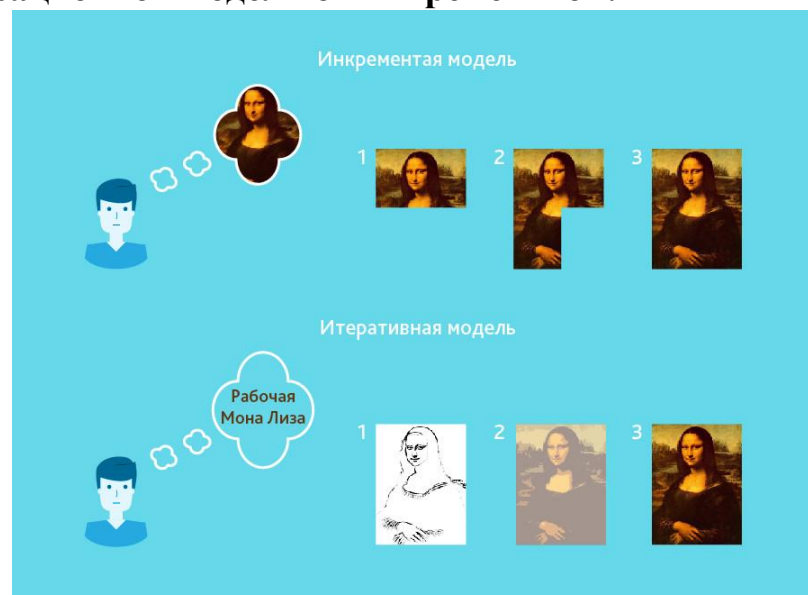
область применения:

- 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.

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

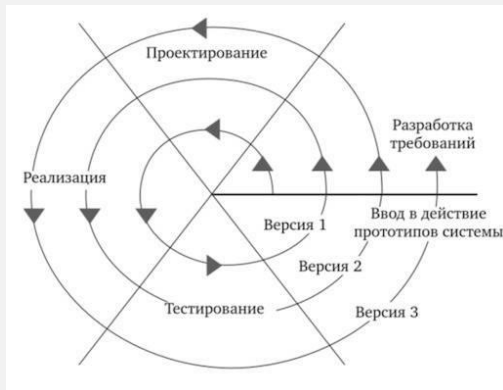


Отличие итерационной модели от инкрементной:



е. Спиральная модель (Spiral Model)

Spiral Model



- 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.

преимущества:

- большое внимание уделяется проработке рисков.

недостатки:

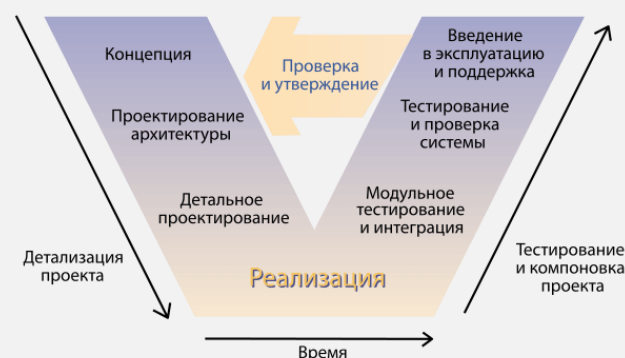
- есть риск застрять на начальном этапе: бесконечно совершенствовать первую версию продукта и не продвинуться к следующим;
- разработка длится долго и стоит дорого.

область применения:

- для решения критически важных бизнес-задач.

ф. V-образная модель (разработка через тестирование)

V-Model (validation and verification)



заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе. Это усовершенствованная **каскадная модель**

преимущества:

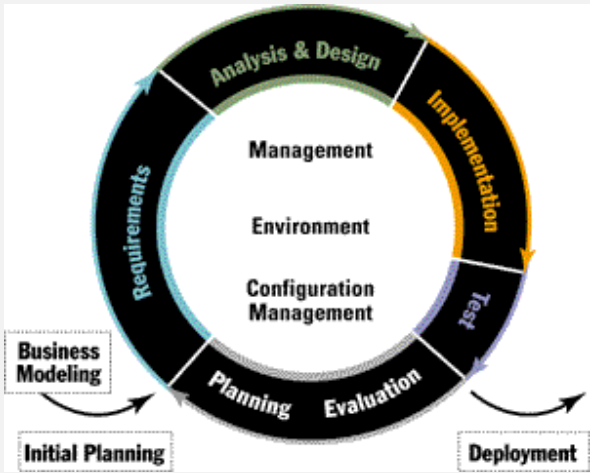
- количество ошибок в архитектуре ПО сводится к минимуму.

недостатки:

- если была допущена ошибка при разработке архитектуры, то

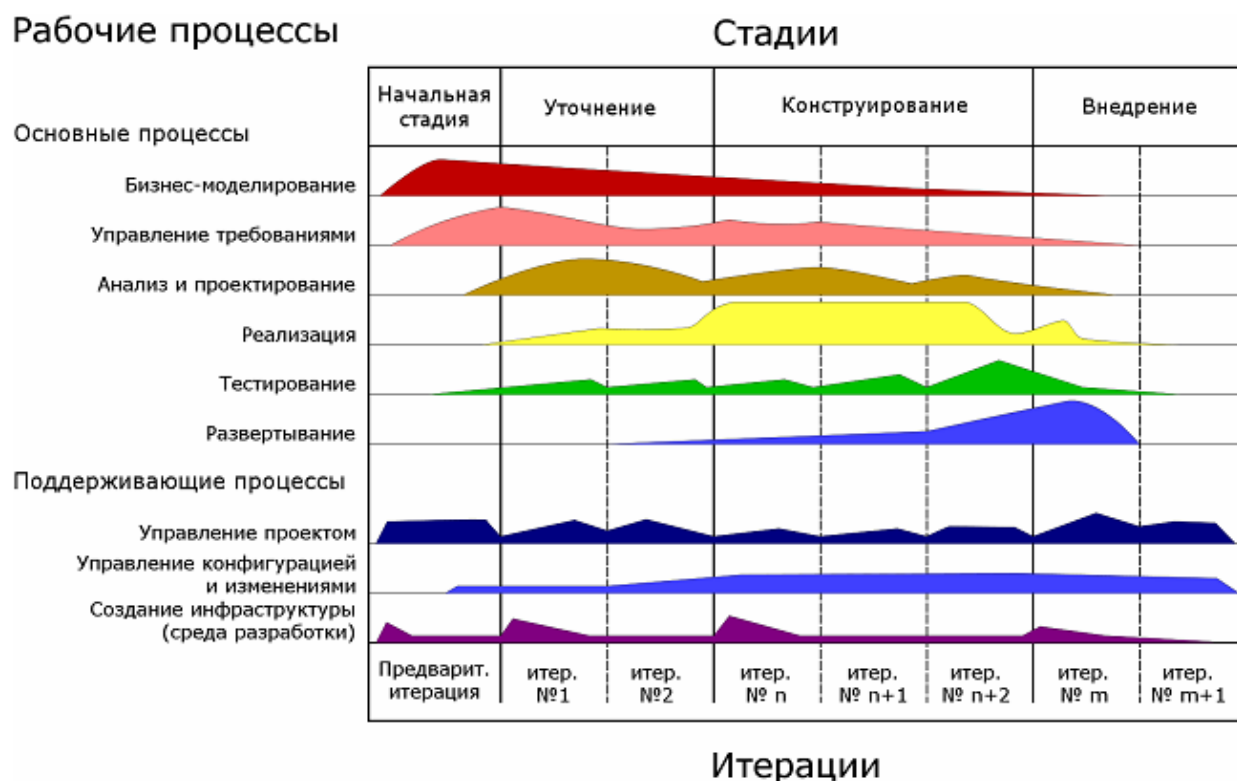
	вернуться и исправить её будет стоить дорого, как и в Waterfall.
<p>область применения:</p> <ul style="list-style-type: none"> – для малых и средних проектов, где требования четко определены и фиксированы; – если требуется тщательное тестирование; – для проектов, в которых важна надёжность и цена ошибки очень высока. 	

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

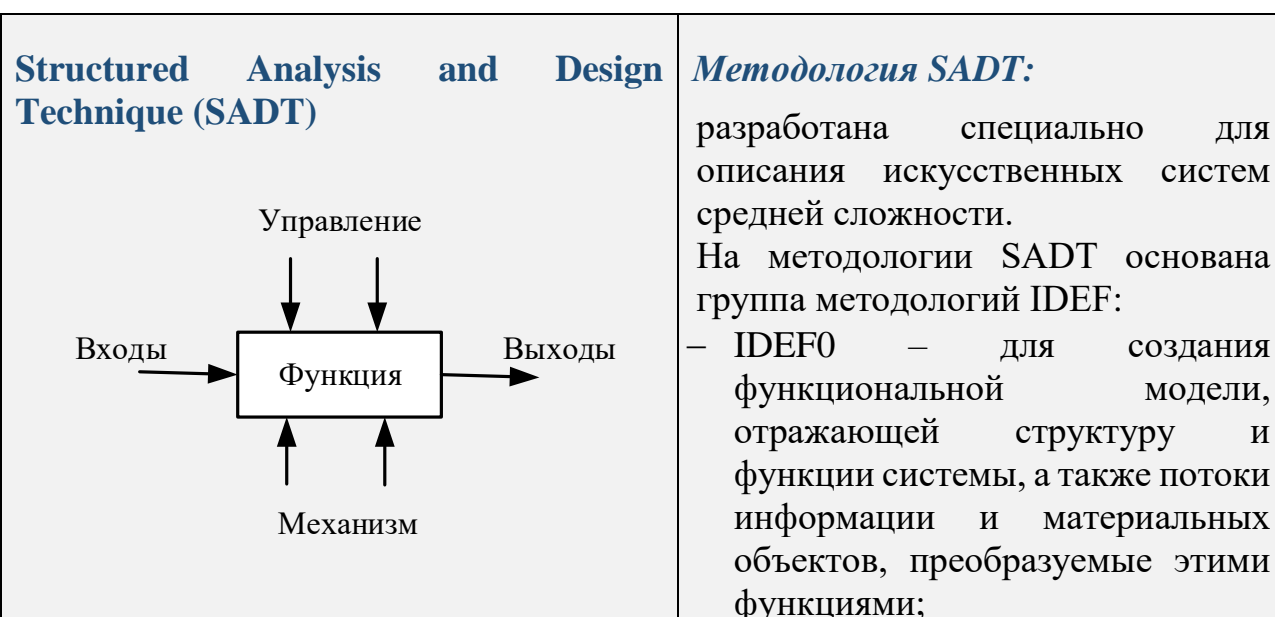
<p>Rational Unified Process (RUP)</p> 	
<p>RUP основные принципы:</p> <ul style="list-style-type: none"> – итеративная модель разработки; – управление требованиями; – компонентная архитектура; – визуальное моделирование ПО; – проверка качества ПО; – контроль внесенных изменений. 	
<p>преимущества:</p> <ul style="list-style-type: none"> – 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. 	<p>недостатки:</p> <ul style="list-style-type: none"> – высокие расходы; – сложность внедрения.
<p>область применения:</p>	

- 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.

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)»



<p>Общее представление процесса (верхний уровень) Процесс (0)</p> <p>Более детальное представление процесса (декомпозиция верхнего уровня) Процесс (0)</p> <p>Более детальное представление одного из подпроцессов верхнего уровня (декомпозиция одного из подпроцессов верхнего уровня) Подпроцесс 2 (2)</p>	<ul style="list-style-type: none"> – 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.
<p>преимущества:</p> <ul style="list-style-type: none"> – 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. 	<p>недостатки:</p> <ul style="list-style-type: none"> – сложность восприятия диаграмм; – большое количество уровней декомпозиции; – трудность увязки нескольких процессов.
<p>область применения:</p> <ul style="list-style-type: none"> – используется при описании банковского дела, организации материально-технического снабжения, построении системы планирования производства. 	

і. методология гибкой разработки ПО (Agile Model)

<p>Agile Model</p> <p>МОДЕЛИ</p> <p>Водопад, Инкрементная, Итеративная, Спиральная, V модель</p> <p>AGILE</p> <p>Методологии</p> <p>XP, Scrum, Kanban, Lean, TDD, FDD, Open UP</p>	<p>Включает в себя:</p> <ol style="list-style-type: none"> 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);
---	---

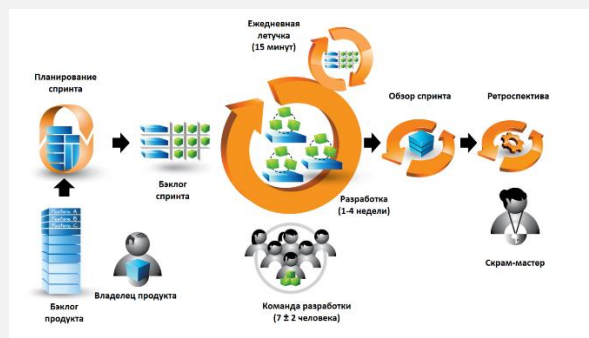
	<ol style="list-style-type: none"> 6. методологию «чистой комнаты» (Cleanroom Software Engineering); 7. итеративно-инкрементальный метод разработки (OpenUP); 8. методологию разработки Microsoft Solutions Framework (MSF); 9. метод разработки динамических систем (Dynamic Systems Development Method, DSDM); 10. метод управления разработкой Kanban.
<p>преимущества:</p> <ul style="list-style-type: none"> – гибкость; – позволяет получить более совершенный продукт, поскольку проверка качества осуществляется по окончании каждого спринта; – быстрый выпуск продукта. 	<p>недостатки:</p> <ul style="list-style-type: none"> – 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.
<p>область применения:</p> <ul style="list-style-type: none"> – для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. 	

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 определении 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 заказчику.

область применения:

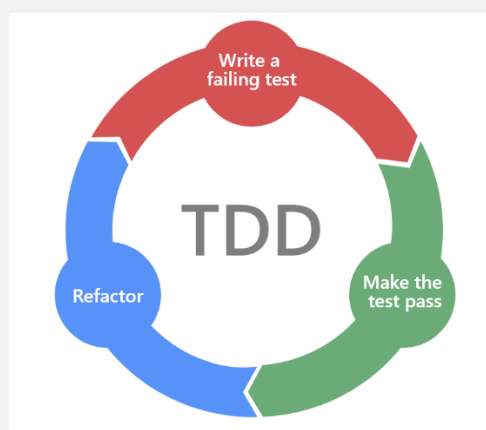
- из-за недостатка структуры и документации не подходит для крупных проектов.

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



TEST DRIVEN DEVELOPMENT

Test-driven development, TDD



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.

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.

недостатки:

- возрастающая сложность для языков с динамической типизацией.

область применения:

- подходит для всех проектов.

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:

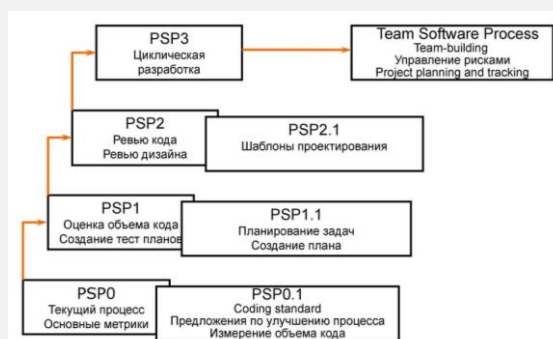
- MSF не описывает детально важнейшие роли заказчика и пользователя;
- не рассматриваются также методы управления группой проектов, из которых состоит основной проект.

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

