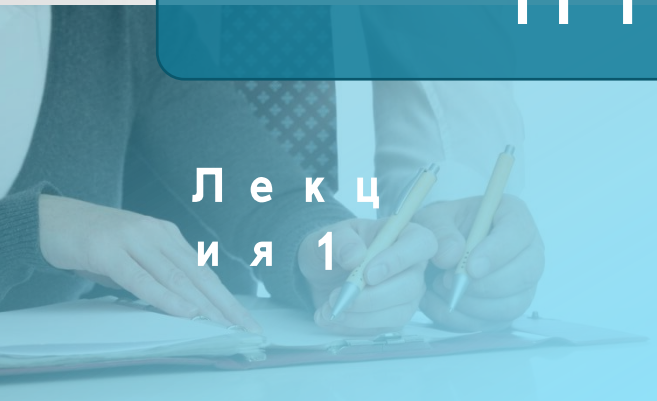
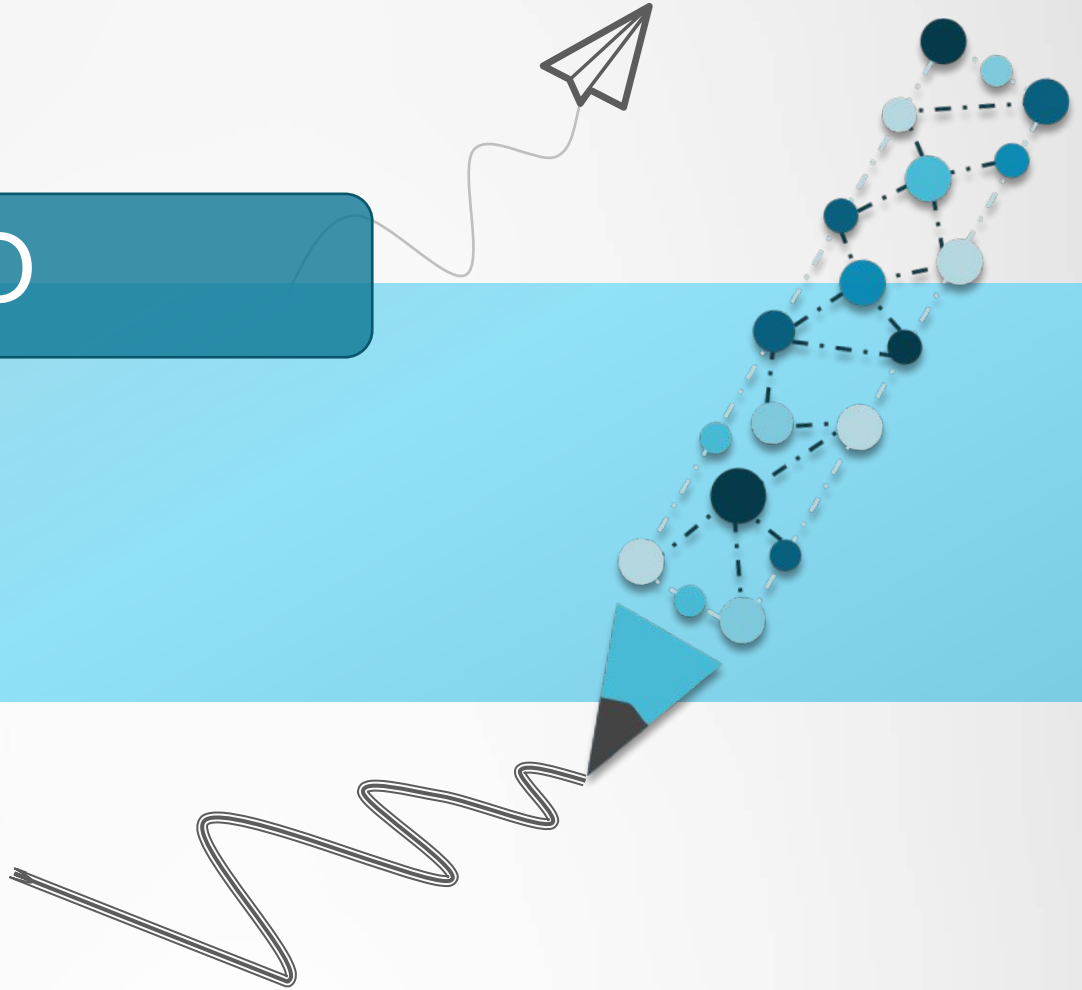
ТРПО

Лекция 1





Начало проекта

Ожидание

Реальность

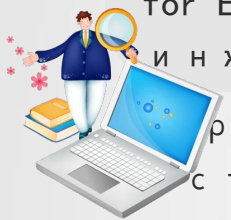
План

- Контекст разработки программного продукта
- Этапы разработки программного обеспечения
- Стратегии разработки
- 10 правил промышленного создания ПО

Контекст разработки программного продукта

Технология разработки программных продуктов — это по определению одна из областей инженерной науки, и поэтому она несет такую же социальную ответственность, как и другие области.

С начала развития компьютерных технологий работу по созданию программных продуктов относили к разработке, для которой требуются в основном навыки программирования, а не знания инженерной науки. Аккредитационный совет по инженерной науке и технологии (ABET — Accreditation Board for Engineering and Technology) определяет профессию инженера следующим образом:

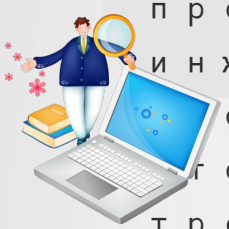


рофессия, в которой математические и стественно-научные знания, полученные исследованиями, опытом и практикой, мудро

Контекст разработки программного продукта

Много труда было вложено в развитие инженерной науки еще до рождения первого программного продукта. Сейчас, в начале третьего тысячелетия, процесс разработки программного продукта начинает требовать от своих создателей такой же высоты научных знаний, как это необходимо в других областях инженерной науки — электронике, механике или строительстве.

В чем сходство и различие между разработкой программного продукта и другими областями инженерной науки? Один общий для них элемент — это необходимость подробного описания того, что должно быть создано, так называемый анализ требований. С другой стороны, программные



Этапы разработки программного обеспечения

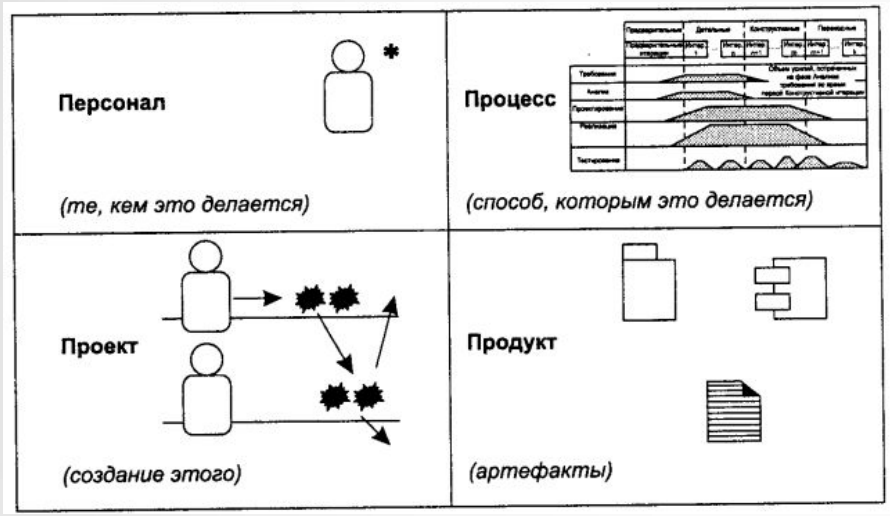
В 80-е и 90-е годы в области разработки программного обеспечения (ПО) преобладали две тенденции. Одна — это быстрый рост приложений, в том числе создаваемых для Web. Другая — это расцвет инструментальных средств и парадигм (подходов к проектированию, таких как объектно-ориентированный).

Однако, несмотря на появление новых тенденций, основные этапы разработки программного обеспечения остались неизменными, как показано ниже.

- Определение процесса разработки ПО, который будет использоваться в дальнейшем.
- Управление проектом разработки.
- Описание целевого программного продукта.
- Проектирование продукта.
- Разработка продукта, то есть его программирование.
- Тестирование частей продукта.
- Интеграция частей и тестирование продукта в целом.
- Сопровождение продукта.

Разработчики меняют последовательность проработки этих направлений и долю внимания, уделяемого каждому направлению. В реальности разработка программного обеспечения обычно определяется требуемым набором функций или сроком сдачи проекта, что диктуется ситуацией на рынке. В результате

Этапы разработки программного обеспечения



Система разработки программного обеспечения включает в себя **персонал, процесс, проект и продукт**.

Использованные обозначения взяты из книги «Унифицированный процесс разработки программного обеспечения» (USDP — Unified Software Development Process) Якобсона, Буча и Рамбо.

USDP — один из процессов для разработки программного

Процесс

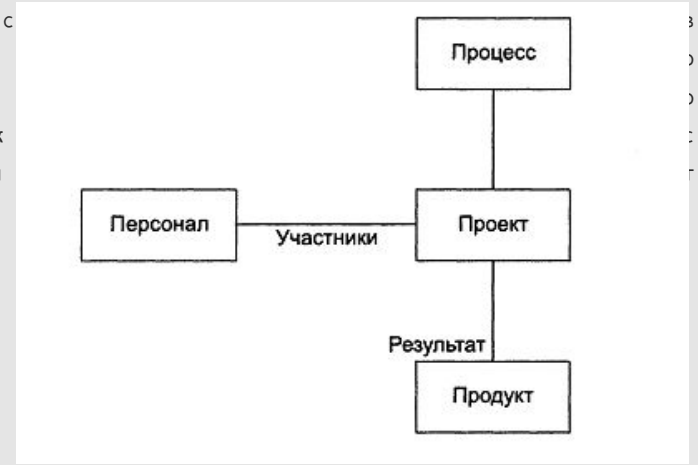
Итак, мы определили, что в основе разработки программного обеспечения лежит проект, который может рассматриваться как организационная сущность, использующаяся для управления разработкой программного обеспечения.

Итогом проекта по разработке программного обеспечения является продукт. Понятие продукта не ограничивается программным кодом. Продукт – это артефакты, создаваемые в течение всей жизни проекта, такие как модели, тексты программ, исполняемые файлы и документация.

Процесс

В создании продукта принимает участие множество различных людей, т.е. персонал. Персонал — это реальные люди, которые участвуют в разработке программного обеспечения в течение всего жизненного цикла. Это архитекторы, разработчики, тестеры, руководители проекта, а также пользователи, заказчики и другие заинтересованные лица.

Усилия людей, вовлеченных в проект, направляет процесс разработки программного обеспечения. Процесс создания программного обес видов деятельности, необходимых для п льзователя в продукт. Процесс, граммного обеспечения, долж сонал. Иначе говоря, людям, работающим ть удобно это делать.



Процесс

Хорошие разработчики программного обеспечения избегают повторения ошибок предыдущих проектов за счет документирования и совершенствования своего **процесса** разработки.

Проектирование программного обеспечения представляет собой процесс построения приложений реальных размеров и практической значимости, удовлетворяющих заданным требованиям функциональности и производительности.

Программирование — это один из видов деятельности, входящих в цикл разработки программного обеспечения.

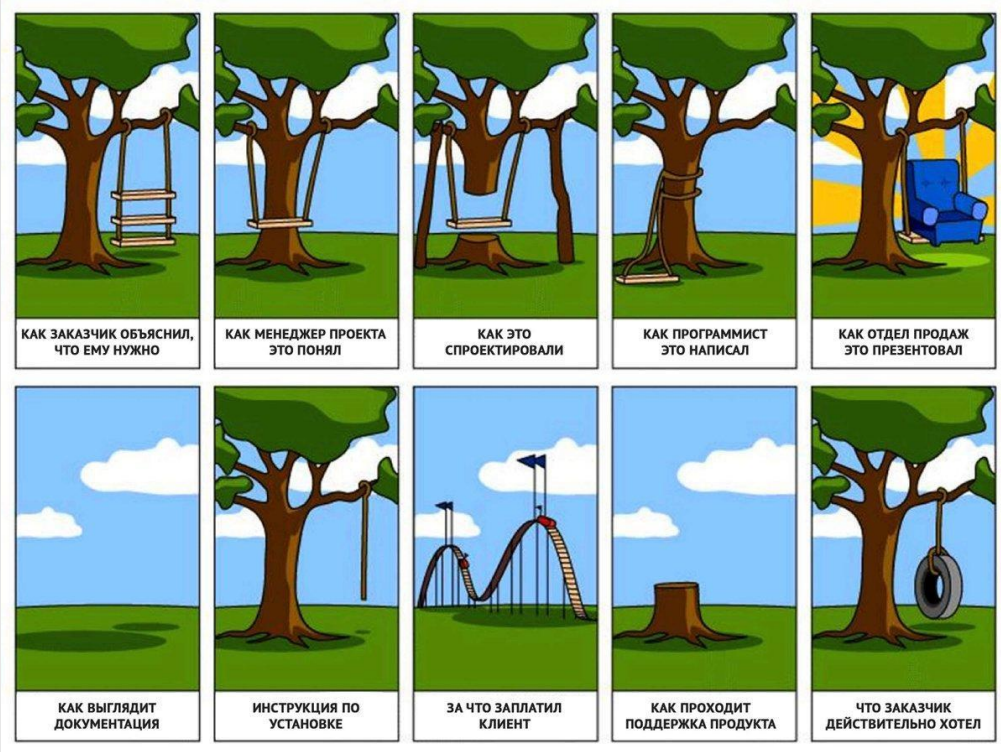
По масштабам работы, требуемым профессиональным знаниям и общественной значимости различие между просто программированием и проектированием программного обеспечения можно сравнить с различием между изготовлением скамейки у ворот своего загородного дома и возведением моста. Эти две задачи различаются на порядок по значимости и требуемым профессиональным знаниям. В отличие от постройки скамейки возведение моста включает в себя как профессиональную, так и циальную ответственность.

Процесс

Типичная схема разработки программного обеспечения:

1. Понять природу и сферу применения предлагаемого продукта.
2. Выбрать процесс разработки и создать план.
3. Собрать требования.
4. Спроектировать и собрать продукт.
5. Выполнить тестирование продукта.
6. Выпустить продукт и обеспечить его сопровождение.

Процесс



1. Понять природу и сферу применения предлагаемого продукта.

Прежде всего, необходимо уяснить суть проекта. Это кажется очевидным, однако для того, чтобы понять, чего хочет заказчик, требуется ощутимое время, особенно если заказчик сам не знает достаточно хорошо, чего он в действительности хочет.

Нужно составить представление о масштабах проекта и с этой целью оценить, какими сроками, финансами и персоналом мы располагаем. Если, например, для построения видеоигры нам предоставляется 10 тыс. долларов, один разработчик и один месяц срока, можно говорить разве что о

Процесс

2. Выбрать процесс разработки и создать план.

С самого начала проекта должна вестись документация, которая, скорее всего, будет изменяться и обновляться в ходе разработки. Поэтому сразу же необходимо определиться и со средствами, при помощи которых будут отслеживаться вносимые в документацию и в программный код изменения. Этот процесс называется управлением конфигурациями. Управление конфигурациями не является абсолютной необходимостью, но его отсутствие может привести к большим недоразумениям и общей потере продуктивности. Далее нужно определиться с самим процессом разработки.

После этого обычно составляется общий план проекта, включающий в себя план-график (расписание проекта). Этот план будет уточняться на протяжении жизненного цикла проекта, по мере того как будут уточняться требования к проекту и детали реализации. Например, детали плана-графика не могут быть проработаны, пока не будет определена архитектура приложения.

Процесс

3. Собрать требования.

Следующий шаг состоит в сборе требований к приложению. Он включает в себя, прежде всего, обсуждение проекта с заказчиками и другими участниками, заинтересованными в его выполнении.

4. Спроектировать и собрать продукт.

Затем наступает черед проектирования и реализации самого продукта.

В зависимости от используемого процесса разработки шаги 3 и 4 могут быть выполнены несколько раз.

5. Выполнить тестирование продукта.

Программный продукт, как окончательный, так и промежуточный, подлежит тщательному тестированию

6. Выпустить продукт и обеспечить его сопровождение.

Наконец, когда продукт выпущен, наступает фаза его сопровождения, включающая внесение в него исправлений и улучшений.

Стратегии разработки

- Каскадная
- V–модель
- Итеративная модель
- Спиральная модель
- Модель быстрой разработки RAD

Каскадная модель

Однажды, на заре эпохи программирования, единственными участниками процесса разработки были программисты. Они писали программные функции для облегчения математических расчетов или автоматизации других рутинных действий. Сегодня все иначе. Современные системы настолько большие и сложные, что над их созданием трудятся целые команды специалистов разного профиля: программисты, аналитики, системные администраторы, тестировщики и конечные пользователи. Все они работают вместе для разработки программ, содержащих миллионы строк кода.

Самой старой и известной моделью построения

Каскадная модель

Каскадная модель проста и понятна, но не так практична как раньше. В условиях динамично изменяющихся требований, строго структурированный процесс может из преимущества превратиться в помеху на пути успешного завершения разработки системы.

Достоинства модели:

- стабильность требований в течение всего жизненного цикла разработки;
- на каждой стадии формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- определенность и понятность шагов модели и простота её применения;
- выполняемые в логической последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие ресурсы (денежные. материальные и людские).

каскадная модель хорошо зарекомендовала себя при построении ительно простых ПО, когда в самом начале разработки можно аточно точно и полно сформулировать все требования к продукту.

Каскадная модель

Недостатки модели:

- ложность чёткого формулирования требований и невозможность их динамического изменения на протяжении пока идет полный жизненный цикл;
- низкая гибкость в управлении проектом;
- последовательность линейной структуры процесса разработки, в результате возврат к предыдущим шагам для решения возникающих проблем приводит к увеличению затрат и нарушению графика работ;
- непригодность промежуточного продукта для использования;
- невозможность гибкого моделирования уникальных систем;
- позднее обнаружение проблем, связанных со сборкой, в связи с одновременной интеграцией всех результатов в конце разработки;
- недостаточное участие пользователя в создании системы — в самом начале (при разработке требований) и в конце (во время приёмочных испытаний);
- пользователи не могут убедиться в качестве разрабатываемого продукта до окончания всего процесса разработки. Они не имеют возможности оценить качество, т.к.нельзя увидеть готовый продукт разработки;
- у пользователя нет возможности постепенно привыкнуть к системе. Процесс обучения происходит в конце жизненного цикла, когда ПО уже запущено в эксплуатацию;
- каждая фаза является предпосылкой для выполнения последующих действий,

Каскадная модель

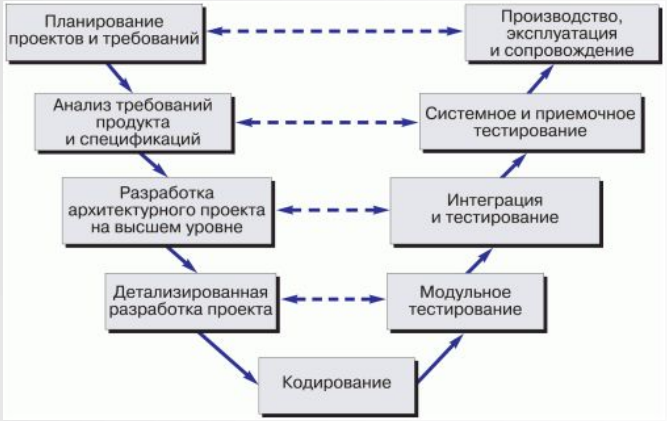
Несмотря на то, что каскадная модель все еще используется, она уже утратила былые позиции. Сегодня ей на смену приходят более продвинутые модели и методологии разработки программного обеспечения.

Область применения Каскадной модели

Ограничение области применения каскадной модели определяется её недостатками. Её использование наиболее эффективно в следующих случаях:

- при разработке проектов с четкими, неизменяемыми в течение жизненного цикла требованиями, понятными реализацией и техническими методиками;
- при разработке проекта, ориентированного на построение системы или продукта такого же типа, как уже разрабатывались разработчиками ранее;
- при разработке проекта, связанного с созданием и выпуском новой версии уже существующего продукта или системы;
- при разработке проекта, связанного с переносом уже существующего продукта или системы на новую платформу;

V-модель



V-модель - это улучшенная версия классической каскадной модели. Здесь на каждом этапе происходит контроль текущего процесса, для того чтобы убедится в возможности перехода на следующий уровень. В этой модели тестирование начинается еще со стадии написания требований, причем для каждого последующего этапа предусмотрен свой уровень тестового покрытия.

Для каждого уровня тестирования разрабатывается отдельный тест-план, то есть во время тестирования текущего уровня, мы также занимаемся разработкой стратегии тестирования следующего. Создавая тест-планы, мы также определяем ожидаемые результаты тестирования и указываем критерии входа и выхода для каждого этапа.

В V-модели каждому этапу проектирования и разработки системы соответствует

V-модель

Плюсы и минусы V-модели:

+ строгая этапизация;

+ планирование тестирования и верификация системы производятся на ранних этапах;

+ улучшенный, по сравнению с каскадной моделью, тайм-менеджмент;

+ промежуточное тестирование.

— недостаточная гибкость модели;

— собственно создание программы происходит на этапе написания кода, то есть уже в середине процесса разработки;

— недостаточный анализ рисков;

— нет работы с параллельными событиями и возможности динамического внесения изменений.

Когда использовать V-модель:

- В проектах, в которых существуют временные и финансовые ограничения;

- Для задач, которые предполагают более широкое, по сравнению с каскадной моделью, тестовое покрытие.

Итеративная модель

Не все модели жизненного цикла последовательны. Существуют также итеративные (или инкрементальные) модели, в которых используется другой подход. Вместо одной продолжительной последовательности действий здесь весь жизненный цикл продукта разбит на ряд отдельных мини-циклов. Причем каждый из них состоит из все тех же базовых стадий модели жизненного цикла. Эти мини-циклы называются итерациями. В каждой из итераций происходит разработка отдельного компонента системы, после чего этот компонент добавляется к уже ранее разработанному функционалу.

Итеративная модель не предполагает полного объема требований для начала работ над продуктом. Разработка программы может начинаться с требований к части функционала, которые могут впоследствии дополняться и

Итеративная модель

В несколько упрощенном виде, итеративная модель состоит из четырех основных стадий, которые повторяются в каждой из итераций (plan-do-check-act):

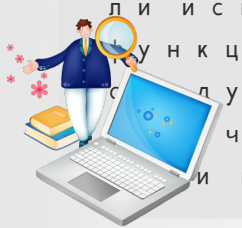
- определение и анализ требований;

- дизайн и проектирование - согласно требованиями. 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 и предложения дополнений к ним.

По результатам каждой итерации принимается решение - будут ли использованы ее результаты для дополнения существующей функциональности в качестве входной точки для начала следующей итерации (т.н. инкрементальное прототипирование). В конечном итоге, достигается точка, в которой все требования и воплощены в продукте - происходит релиз.



Итеративная модель

В математических терминах, итеративная модель представляет реализацию методики последовательной аппроксимации – то есть, постепенное приближение к образу готового продукта:

Ключ к успешному использованию этой модели – строгая валидация требований и тщательная верификация разрабатываемой функциональности в каждой из итераций.

Основные стадии процесса разработки в итеративной модели фактически повторяют модель водопада. В каждой итерации создается программное обеспечение, требующее тестирования на всех уровнях.

Итеративная модель

Достоинства:

- затраты, которые получаются в связи с изменением требований пользователей, уменьшаются, повторный анализ и совокупность документации значительно сокращаются по сравнению с каскадной моделью;
- легче получить отзывы от клиента о проделанной работе — клиенты могут озвучить свои комментарии в отношении готовых частей и могут видеть, что уже сделано. Т.к. первые части системы являются прототипом системы в целом.
- у клиента есть возможность быстро получить и освоить программное обеспечение — клиенты могут получить реальные преимущества от системы раньше, чем это было бы возможно с каскадной моделью.

Недостатки модели:

- менеджеры должны постоянно измерять прогресс процесса. в случае быстрой разработки не стоит создавать документы для каждого минимального изменения версии;
- структура системы имеет тенденцию к ухудшению при добавлении новых компонентов — постоянные изменения нарушают структуру системы. Чтобы избежать этого требуется дополнительное время и

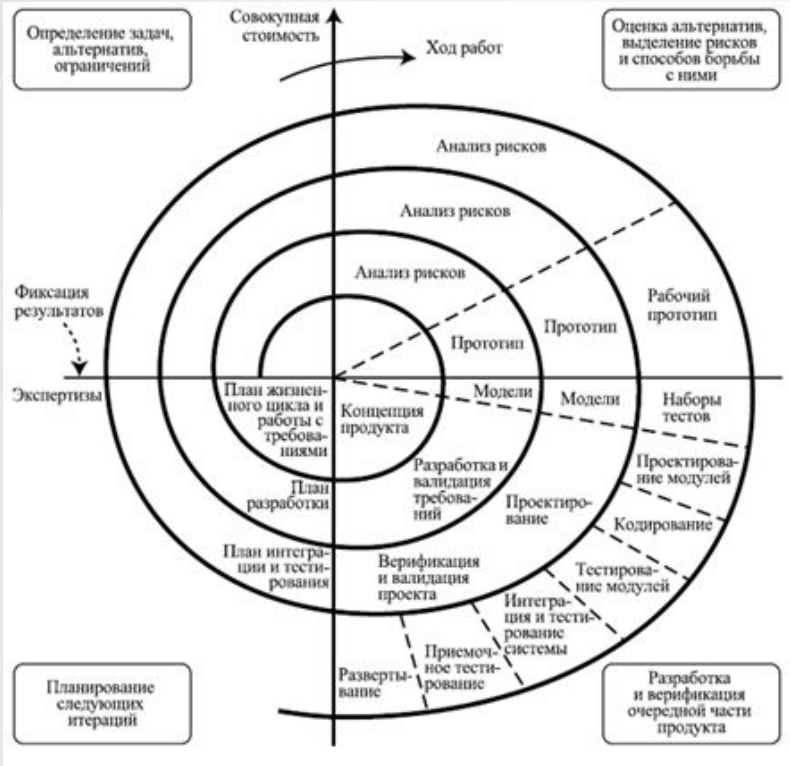
Итеративная модель

Когда использовать итеративную модель:

- для крупных проектов;
- когда известны, по крайней мере, ключевые требования;
- когда требования к проекту могут меняться в процессе разработки.

Итеративная модель является ключевым элементом так называемых «гибких» (Agile) подходов к разработке программного обеспечения, основные из которых мы рассмотрим в следующих разделах.

Спиральная модель



Спиральная модель представляет шаблон процесса разработки ПО, который сочетает идеи итеративной и каскадной моделей. Суть ее в том, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали, которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны

Спиральная модель

Данная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипировнаие с целью сочетания преимуществ восходящей и нисходящей концепции, делающая упор на начальные этапы жизненного цикла: анализ и проектирование. Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла.

На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, планировать работы следующего витка спирали. Таким образом уточняются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который удовлетворяет действительным требованиям заказчика и доводится до реализации.

Спиральная модель

Жизненный цикл на каждом витке спирали — могут применяться разные модели процесса разработки ПО. В конечном итоге на выходе получается готовый продукт.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем.

Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Спиральная модель

Достоинства модели:

- позволяет быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований;
- допускает изменение требований при разработке программного обеспечения, что характерно для большинства разработок, в том числе и типовых;
- в модели предусмотрена возможность гибкого проектирования, поскольку в ней воплощены преимущества каскадной модели, и в то же время разрешены итерации по всем фазам этой же модели;
- позволяет получить более надежную и устойчивую систему. По мере развития программного обеспечения ошибки и слабые места обнаруживаются и исправляются на каждой итерации;
- эта модель разрешает пользователям активно принимать участие при планировании, анализе рисков, разработке, а также при выполнении оценочных действий;
- уменьшаются риски заказчика. Заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта;
- обратная связь по направлению от пользователей к разработчикам

Спиральная модель

Недостатки модели:

- если проект имеет низкую степень риска или небольшие размеры, модель может оказаться дорогостоящей. Оценка рисков после прохождения каждой спирали связана с большими затратами;
- жизненный цикл модели имеет усложненную структуру, поэтому может быть затруднено её применение разработчиками, менеджерами и заказчиками;
- спираль может продолжаться до бесконечности, поскольку каждая ответная реакция заказчика на созданную версию может порождать новый цикл, что отдаляет окончание работы над проектом;
- большое количество промежуточных циклов может привести к необходимости в обработке дополнительной документации;
- использование модели может оказаться дорогостоящим и даже недопустимым по средствам, т.к. время. затраченное на планирование, повторное определение целей, выполнение анализа рисков и прототипирование, может быть чрезмерным;
- могут возникнуть затруднения при определении целей и стадий, указывающих на готовность продолжать процесс разработки на следующей итерации

Спиральная модель

Основная проблема спирального цикла — определение момента перехода на следующий этап. Для её решения вводятся временные ограничения на каждый из этапов жизненного цикла и переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. Планирование производится на основе статистических данных, полученных в предыдущих проектах и личного опыта разработчиков.

Область применения спиральной модели

Применение спиральной модели целесообразно в следующих случаях:

- при разработке проектов, использующих новые технологии;
- при разработке новой серии продуктов или систем;
- при разработке проектов с ожидаемыми существенными изменениями или дополнениями требований;
- для выполнения долгосрочных проектов;
- при разработке проектов, требующих демонстрации качества и версий системы или продукта через короткий период времени;
- при разработке проектов. для которых необходим подсчет затрат, связанных с оценкой и разрешением рисков.

Модель быстрой разработки RAD

Модель быстрой разработки приложений (Rapid Application Development, RAD) появилась в 80-е годы прошлого века в связи с бурным развитием инструментальных средств разработки программных продуктов. Концепцию RAD часто связывают с концепцией визуального программирования. Поэтому в процессе быстрой разработки приложений основное внимание уделяется не программированию и тестированию, а анализу требований и проектированию.

Данная модель, исходя из особенностей ее реализации и целей ее использования, может поддерживать как инкрементную, так и эволюционную стратегию разработки программного обеспечения. Как правило, RAD-модели используются в составе другой модели для ускорения цикла разработки прототипа системы или программного средства. При невысокой сложности проектов RAD-модели могут применяться как независимые.

Модель быстрой разработки RAD

Характерной чертой RAD-модели является короткое время перехода от анализа требований до создания полной системы или программного средства. Подход RAD предполагает наличие:

- небольших групп профессиональных разработчиков (до 7 чел.), выполняющих работы по проектированию отдельных подсистем программного обеспечения;
- хорошо проработанного производственного графика;
- циклической реализации в программном продукте требований, получаемых в результате взаимодействия с заказчиком.

Модель быстрой разработки RAD

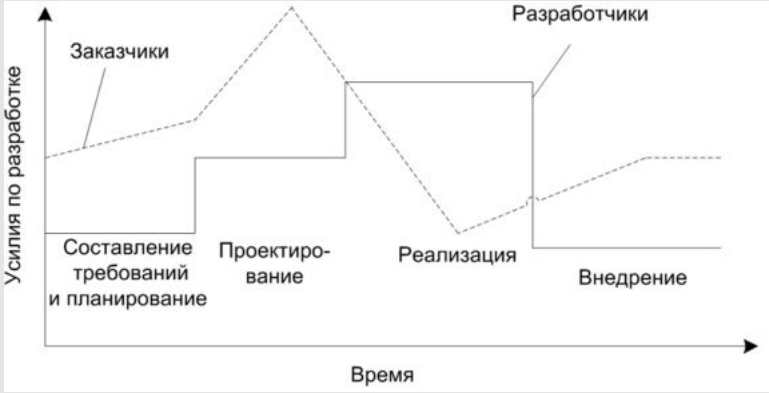
Жизненный цикл программного обеспечения в соответствии с подходом RAD включает четыре стадии, такие как:

1) анализ и формирование требований;

2) проектирование;

3) реализация;

4) внедрение и сопровождение.



Модель быстрой разработки RAD

На стадии **анализа и формирования требований**:

- определяют функции, которые должна выполнять система;
- выделяют наиболее приоритетные функции, требующие проработки в первую очередь;
- описывают информационные потребности.

Формулирование требований к системе осуществляется в основном совместными усилиями пользователей и специалистов-разработчиков. Кроме того, на данной стадии:

- устанавливаются границы проекта;
- определяются временные рамки для каждой из последующих стадий;
- решается сама возможность реализации проекта в заданных размерах финансирования, с имеющимся техническим обеспечением и т.п.

В результате завершения стадии должен быть сформулирован перечень функций будущего программного

Модель быстрой разработки RAD

На стадии **проектирования** часть пользователей также принимает участие в проектировании системы под руководством специалистов-разработчиков.

Для быстрого получения работающих прототипов приложений используются соответствующие инструментальные средства — **CASE-средства***. Пользователи, непосредственно взаимодействуя с разработчиками, уточняют и дополняют требования в техническом проектировании к системе, которые не были выявлены на предыдущей стадии.

На данной стадии более детально рассматриваются процессы системы, создаются частичные прототипы (экранная форма, диалог, отчет), определяется состав необходимой документации.

***CASE средства (Computer – Aided Software Engineering)** – это инструмент, который позволяет автоматизировать процесс разработки информационной системы и программного обеспечения. Разработка и создание информационных систем управления предприятием связаны с выделением бизнес-процессов, их

Модель быстрой разработки RAD

На стадии **реализации** непосредственно происходит быстрая разработка приложения. Разработчики производят итеративное построение программного обеспечения на основе полученных на предыдущей стадии моделей, а также требований к надежности, производительности и т.п.

Пользователи оценивают получаемые результаты и вносят коррективы, если в процессе разработки ПО перестает удовлетворять определенным ранее требованиям. Тестирование программного обеспечения осуществляется в процессе разработки.

Результатом стадии является готовый программный продукт, удовлетворяющий всем согласованным требованиям.

Модель быстрой разработки RAD

На стадии **внедрения** производятся обучение пользователей, организационные изменения и параллельно с внедрением нового программного продукта продолжается эксплуатация существующего до полного внедрения нового.

Планирование и подготовка к внедрению должны начинаться на стадии **проектирования** программного продукта.

Модель быстрой разработки RAD

Следует заметить, что подход RAD не может претендовать на универсальность. Он хорош в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика, и заказчик может принять непосредственное участие в процессе разработки. Если же разрабатывается крупномасштабная программная система, то для такого проекта необходимы высокий уровень планирования, жесткое проектирование, строгое следование заранее разработанным протоколам, что снижает скорость разработки.

Подход RAD не применим также для построения сложных расчетных программ, операционных систем или программ управления сложными объектами в реальном масштабе времени, т.е. программ, содержащих большой объем кода. Быстрая разработка не используется для приложений, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход предполагает, что первые несколько

Модель быстрой разработки RAD

При использовании RAD-модели в соответствующем ей проекте проявляются следующие ее достоинства:

- сокращение продолжительности цикла разработки и всего проекта в целом;
- сокращение количества разработчиков;
- вследствие предыдущих факторов — снижение стоимости проекта (также и за счет использования мощных инструментальных средств);
- сокращение риска несоблюдения графика;
- сокращение риска, связанного с неудовлетворенностью заказчика полученным программным продуктом, за счет привлечения его к циклу разработки;
- возможность повторного использования разработанных компонентов; это достоинство проявляется при использовании RAD-модели в составе инкрементной или эволюционной модели. В этом случае наращивание функциональных возможностей осуществляется на базе

Модель быстрой разработки RAD

Основными недостатками RAD-модели при использовании в неподходящем для нее проекте являются:

- необходимость постоянного участия пользователей в процессе разработки, что не всегда выполнимо и может повлиять на качество конечного продукта;
- жесткость временных ограничений на разработку прототипа;
- сложность определения и ограничения затрат и сроков завершения работы над продуктом.

Модель быстрой разработки RAD

RAD–модель может эффективно применяться в следующих случаях:

- при разработке программных продуктов, если они поддаются моделированию, являются некритическими, имеют небольшой размер и низкую производительность, относятся к известной разработчикам предметной области, являются информационными системами;
- требования для программного обеспечения хорошо известны;
- имеются пригодные к повторному использованию в разрабатываемом ПО компоненты;
- пользователь может принимать постоянное участие в процессе разработки;
- в проекте заняты разработчики, обладающие достаточными навыками в использовании инструментальных средств разработки;
- при разработке ПП, для которых требуется быстрое

10 правил промышленного создания ПО

10 правил промышленного создания ПО или принципы традиционного управления программными проектами

В 1988 году вышла статья Барри Боэма «Список 10 правил промышленного создания ПО», которые наиболее полно отражают основные принципы традиционного управления программными проектами.

1. Поиск и обнаружение ошибки в ПО после его сдачи в эксплуатацию обходится в 100 раз дороже, чем поиск и обнаружение ошибки на ранних стадиях разработки.

Эта оценка не является уникальной для разработки ПО. По отношению к любому продукту материального производства вполне справедливо можно утверждать, что исправление дефекта, обнаруженного после продажи, может стоить намного дороже, чем выявление дефекта на стадии разработки или производства.

2. Можно сократить срок разработки ПО на 25% от номинального, но не более.

Чтобы понять причину этого ограничения, следует вспомнить утверждение Ф. Брукса о том, что для сокращения времени разработки на n% требуется увеличить количество персонала на m% (в предположении, что другие параметры остаются неизменными). Т.е. любой рост числа работников требует увеличения затрат на управление. Оптимальным для

10 правил промышленного создания ПО

3. На каждый доллар, вложенный в разработку, приходится тратить два доллара на сопровождение.

Боэм назвал это правило «железным законом разработки ПО». Независимо от того, создается ли продукт долговременного использования, у которого выходят по две коммерческие версии в год, или единственное в своем роде ПО для конкретного заказчика, количество денег, затрачиваемых на весь цикл сопровождения, будет с некоторой вероятностью в два раза больше суммы, истраченной за весь цикл разработки. На первый взгляд трудно понять, хорошее это соотношение или плохое. Для сектора коммерческих продуктов оно определяется, прежде всего, коммерческим успехом на рынке. Успешные программные продукты (такие, как Oracle, приложения Microsoft, Rational Rose или операционная система UNIX) существуют в течение очень долгого времени, что может приводить к более высокому значению отношения стоимости сопровождения к стоимости разработки. С другой стороны, менеджеры проектов по созданию «одноразового» ПО редко планируют большие затраты на его сопровождение. В любом случае каждому, работает в IT-индустрии следует помнить, что ПО всегда трудно сопровождать.

4. Стоимость разработки и сопровождения программного обеспечения является, прежде всего, функцией числа строк исходного кода.

Это правило справедливо по отношению к заказному ПО, которое

10 правил промышленного создания ПО

5. Различия в уровне квалификации разработчиков приводят к огромной разнице в продуктивности при создании программного обеспечения.

Это самая главная мудрость традиционного процесса: нанимайте хороших работников. Данному правилу зачастую придается как недостаточное, так и чрезмерное значение. Когда отсутствуют объективные сведения относительно причин успеха или неудачи, очевидным козлом отпущения становится квалификация персонала. Это суждение субъективно, и его трудно оспаривать.

6. Общее отношение стоимости программного обеспечения к стоимости аппаратного обеспечения продолжает расти. В 1955 г. оно составляло 15:85; в 1985 г. – 85:15.

Тот факт, что на ПО приходится 85% стоимости большинства систем, является не столько следствием продуктивности создания ПО, сколько следствием того уровня функциональных задач, выполнение которых возлагается на программное обеспечение при принятии решений о системе в целом. Потребность в программном обеспечении, широта его использования и его сложность продолжают безгранично расти.

10 правил промышленного создания ПО

7. При создании ПО всего лишь около 15% усилий затрачивается собственно на программирование.

Для успешного осуществления проекта приходится помимо кодирования выполнять много других задач. Управление требованиями, проектирование, тестирование, планирование, контроль за ходом проекта, управление изменениями — одинаково важные задачи, на которые тратится приблизительно 85% ресурсов.

8. Программные системы и продукты обычно стоят в три раза дороже в пересчете на одну строку исходного кода, чем отдельные программы. Продукты, состоящие из программных систем (т.е. системы систем), стоят дороже в девять раз.

Эта экспоненциальная зависимость лежит в основе того, что называют платой за масштаб (diseconomy of scale). В отличие от других товаров, чем больше объем создаваемого программного обеспечения, тем дороже оно обходится в пересчете на одну строку исходного кода.

10 правил промышленного создания ПО

9. Сквозной контроль позволяет обнаружить 60% ошибок.

Сквозной контроль и другие формы контроля, производимого человеком, хороши для обнаружения ошибок, лежащих на поверхности, и проблем, касающихся стиля программирования, но он не позволяет обнаруживать проблемы второго, третьего, n-ого порядка, такие как конфликты ресурсов, узкие места, связанные с производительностью конфликты управления и т.п. Более того, очень немногие люди способны находить даже семантические ошибки первого порядка в тексте программы. Скольким программистам удается компилировать свою собственную программу с первого раза?

10. 80% работы выполняют 20% работающих.

Это правило можно расширить. Следующие фундаментальные постулаты лежат в основе объяснения современного подхода к процессу управления созданием ПО:

80% разработки выполняется для удовлетворения 20% требований;

80% процентов стоимости ПО приходится на 20% компонентов;

80% ошибок возникают по вине 20% компонентов;

80% ПО выбрасывается и заново переделывается из-за 20% ошибок;

80% ресурсов расходуются на 20% компонентов;

80% разработок выполняются с помощью 20% инструментария;

80% успеха обеспечивают 20% людей.

Эти соотношения дают хорошую точку отсчета для поиска способов