Модели жизненного цикла разработки программного обеспечения

Графов Павел,
Литвинов Антон,
Бутов Максим.

Ноябрь 2007

В истории технологии программирования можно выделить три этапа:

- Осмысление опыта разработки больших систем.
- Разработка новых технологических подходов
- Принятие стандартов на состав процессов жизненного цикла программного обеспечения

Классификация технологических подходов

- *Подходы со слабой формализацией.*

(code & fix)

- *Строгие (классические, жесткие, предсказуемые) подходы.*

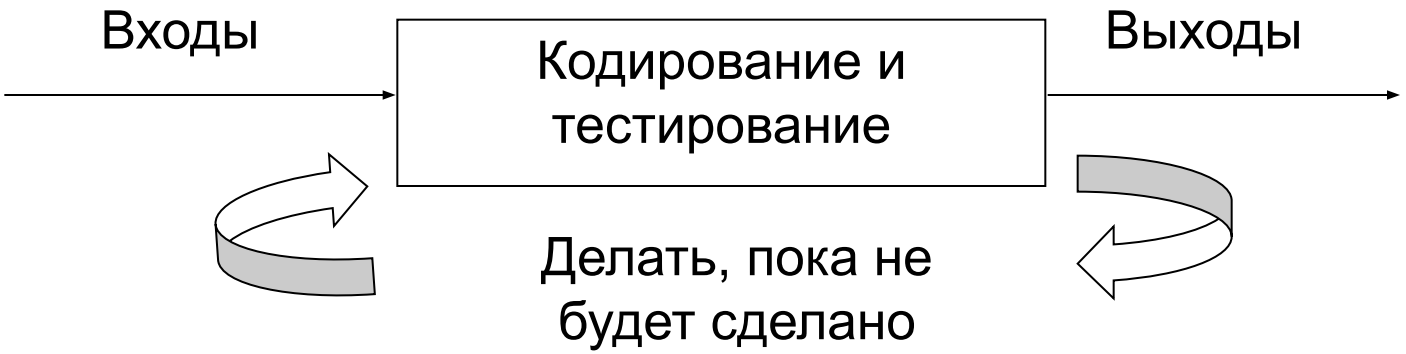
(waterfall model, spiral model, etc.)

- *Гибкие (адаптивные, легкие) подходы.*

(evolutionary prototyping, iterative delivery, extreme programming, etc.)

Code & fix

(подход со слабой формализацией)

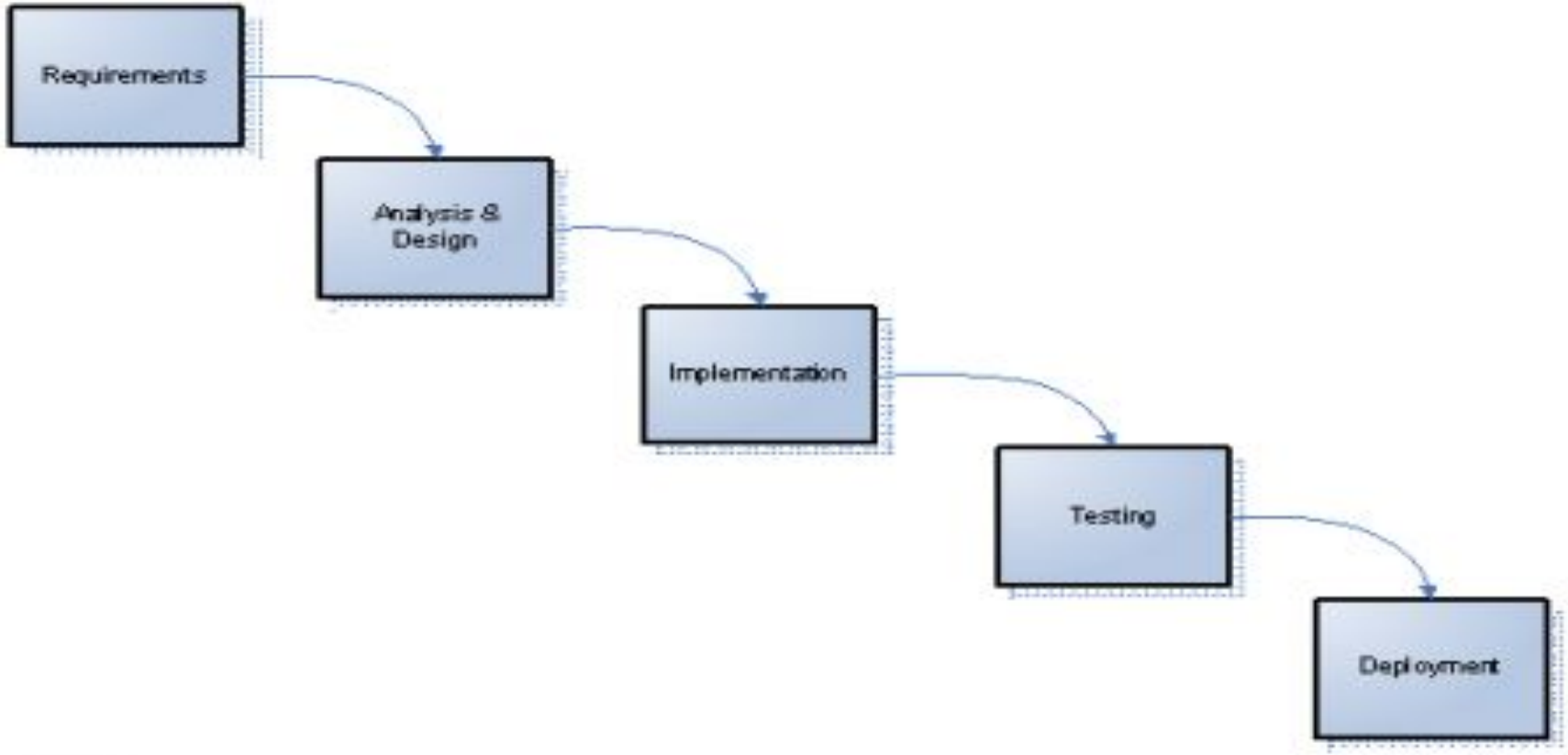


\+ нет затрат времени на проектирование
\- ошибки требуют повторного кодирования

Как правило, это **учебные** или **маленькие несложные** проекты

WaTerFall model

(каскадные технологические подходы – русск.)



анализ->проектирование->программирование->тестирование->сопровождение

Каскадный подход

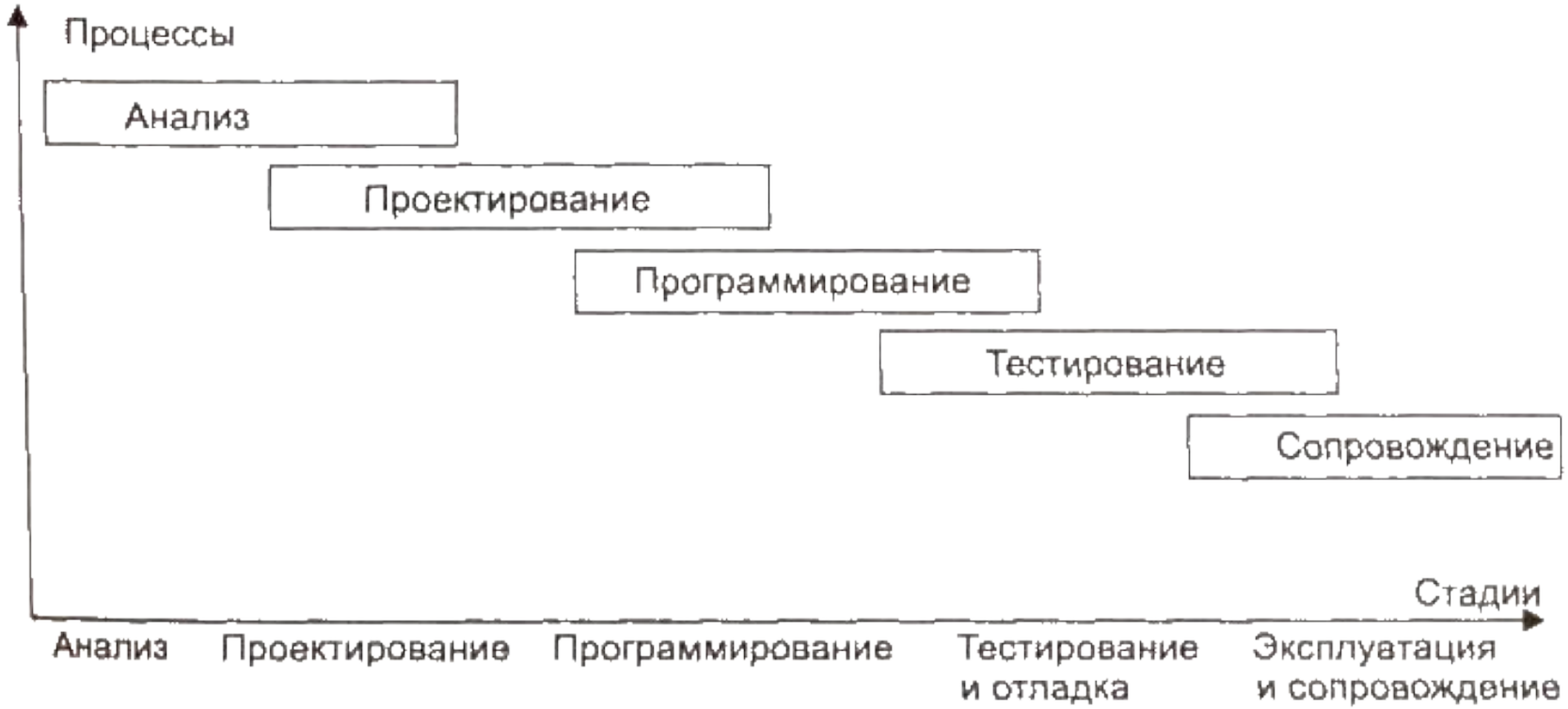
Его достоинства:

- модель доступна для понимания, проста и удобна;
- легко осуществлять контроль;

Его недостатки:

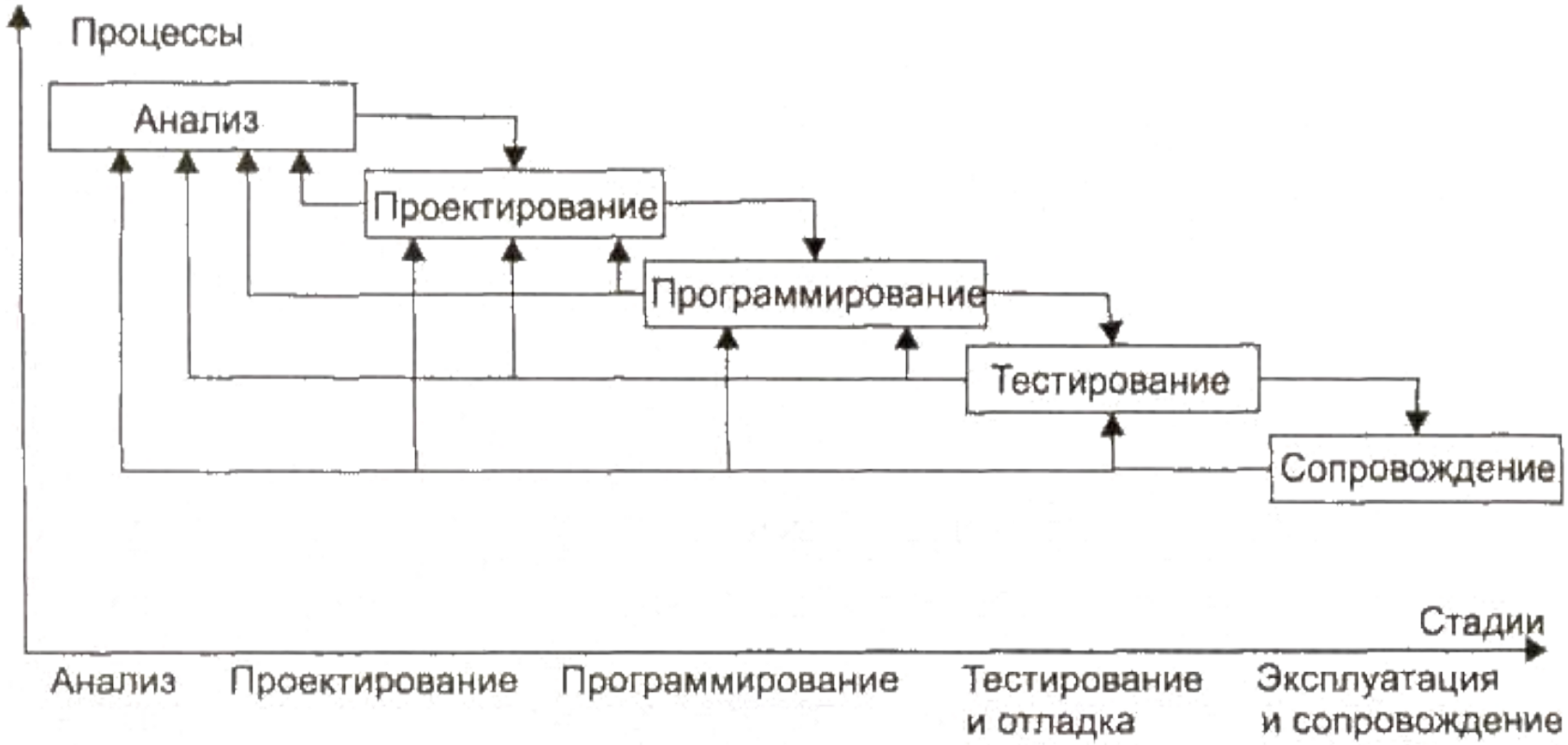
- линейная структура => возврат на одну или несколько фаз назад приводит к удорожанию проекта;
- пользователь принимает участие в процессе разработки только в самом начале — при сборе требований, и в конце — во время приемочных испытаний;
- может создать ошибочное впечатление о работе над проектом

Разновидности каскадной модели



Каскадный подход с перекрывающимися процессами

waterfall with overlapping



Каскадно-возвратный технологический подход

Каскадно-возвратный подход

V- образная модель

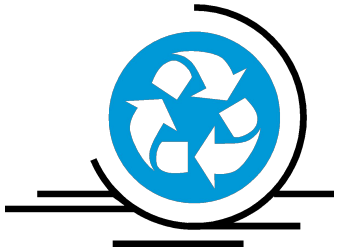
V – образная модель демонстрирует комплексный подход к определению фаз процесса разработки ПО. В ней подчеркнуты взаимосвязи, существующие между аналитическими фазами и фазами проектирования, которые предшествуют кодированию, после которого следуют фазы тестирования. Пунктирные линии означают, что эти фазы необходимо рассматривать параллельно.

Преимущества:

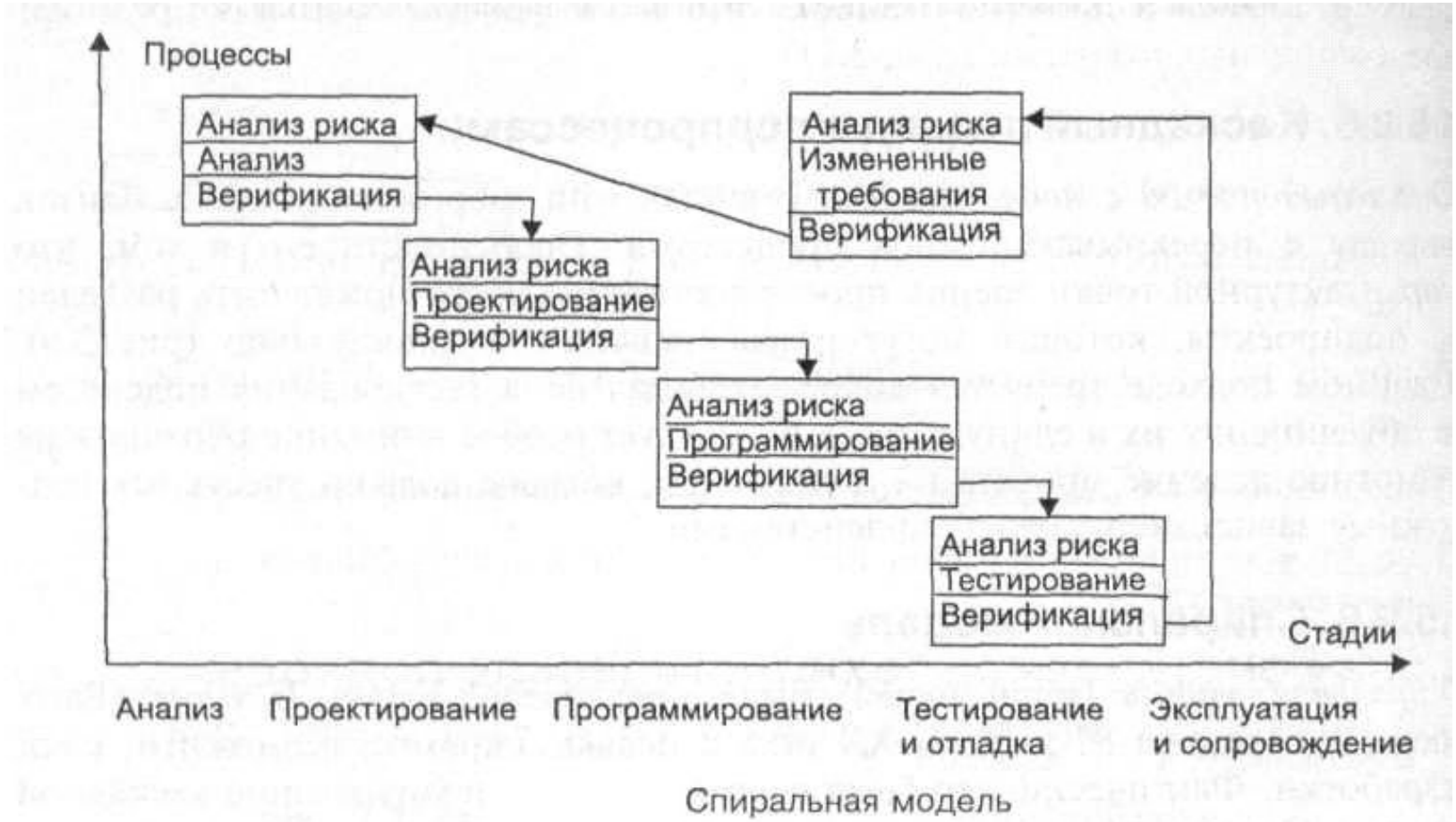
- в модели предусмотрены аттестация и верификация всех внешних и внутренних полученных данных, а не только самого программного продукта;
- в V-образной модели определение требований выполняется перед разработкой проекта системы, а проектирование ПО — перед разработкой компонентов;
- модель определяет продукты, которые должны быть получены в результате процесса разработки, причем каждые полученные данные должны подвергаться тестированию;
- благодаря модели менеджеры проекта может отслеживать ход процесса разработки, так как в данном случае вполне возможно воспользоваться временной шкалой, а завершение каждой фазы является контрольной точкой;
- модель проста в использовании (относительно проекта, для которого она является приемлемом).

Недостатки:

- с ее помощью непросто справиться с параллельными событиями;
- в ней не учтены итерации между фазами;
- в модели не предусмотрено внесение требования динамических изменений на разных этапах жизненного цикла;
- в модель не входят действия, направленные на анализ рисков.



Спиральная модель



Спиральная модель

Эволюционная модель

Идея данной модели состоит в следующем:

1)Разработка первоначальной версии программного продукта, которая передаётся на испытание пользователям.

2)Доработка программного продукта с учётом мнения пользователей после чего получается промежуточная версия продукта, которая также проходит испытание пользователем.

3)Повторная доработка продукта, которая может происходить несколько раз, до тех пор, пока не будет получен необходимый программный продукт.

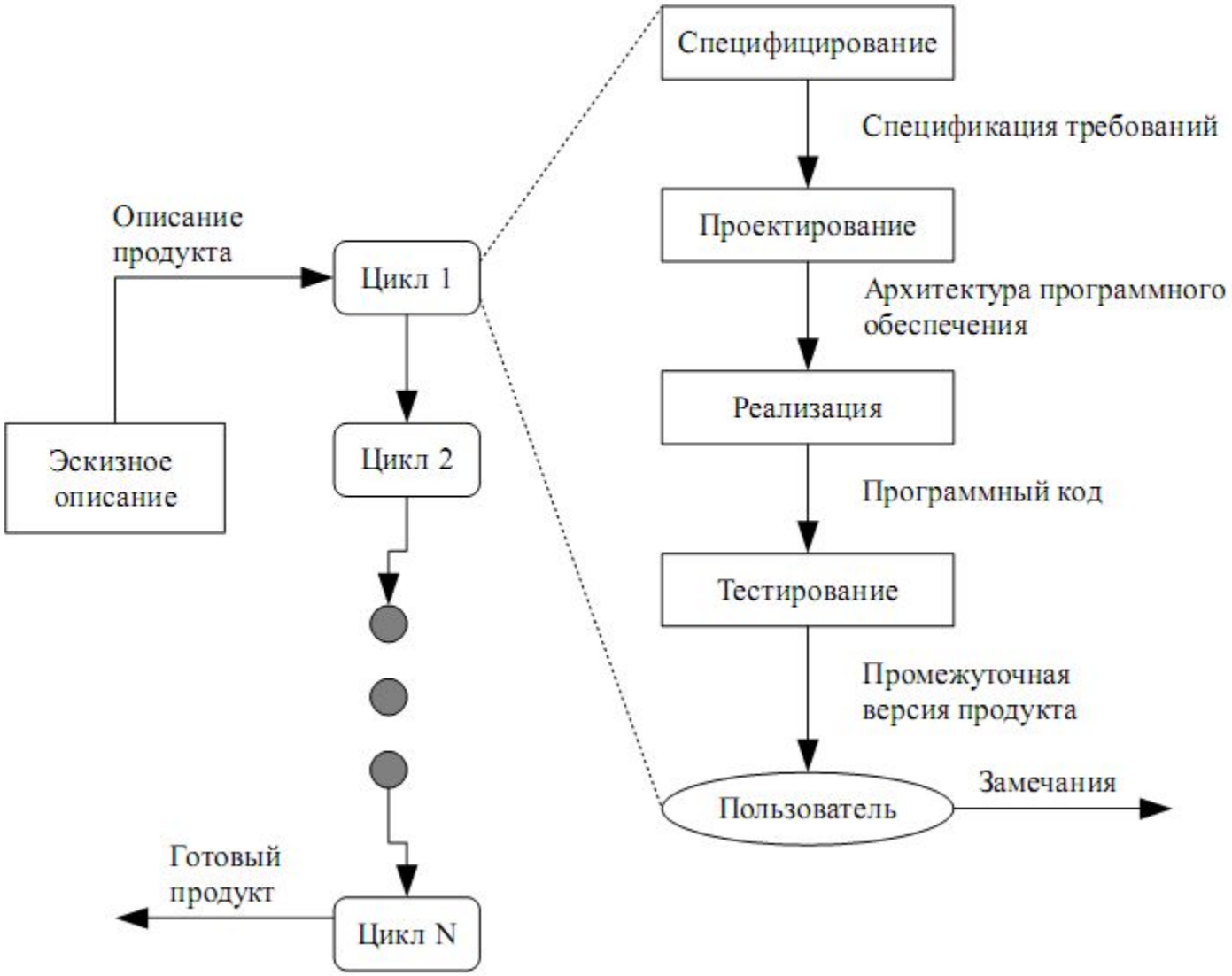
Отличительной чертой данной модели является очень тесное взаимодействие с клиентом.

Различают два подхода к реализации эволюционной модели разработки ПО:

1)Подход пробных разработок

2)Прототипирование

Эволюционная модель



Эволюционная модель

Достоинства эволюционной модели:

1) Более эффективен, чем подход, построенный на основе каскадной модели.
2) Спецификация может разрабатываться постепенно, по мере того как заказчик сформулирует те задачи, которые должно решать ПО.
3) Значительное снижение риска для проекта.
4) Конечный продукт удовлетворяет потребности заказчика и рынка в большей степени.

Недостатки модели:

1) Многие этапы процесса создания ПО не документированы.
2) Система часто получается плохо структурированной.
3) Часто требуются специальные средства и технологии разработки ПО.

Эволюционная модель

Эволюционная модель наиболее приемлема для разработки небольших программных систем (до 100000 строк кода) и систем среднего размера (до 500000 строк кода) с относительно коротким сроком жизни.

Для больших долгоживущих систем рекомендуется использовать смешанный подход. Например:

1) Для прояснения труднопонимаемых мест в системной спецификации и проектирования пользовательского интерфейса можно использовать прототипирование.
2) Часть системных компонентов, для которых полностью определены требования, может создаваться на основе каскадной модели.

Модель «Чистой комнаты»

Данная модель представляет процесс разработки ПО не как метод программных проб и ошибок, а как инженерный процесс с математическим обоснованием.

Она применяет технологии, основанные на теории, такие как:

1) Спецификация типа, ящичная структура, функций использования и архитектуры системных объектов.
2) Функционально-теоретическое проектирование и проверка корректности.
3) Статистическое тестирование использования для аттестации качества.

Модель «Чистой комнаты» основана на инкрементной разработке и аттестации процессов реализации функциональной и пользовательской спецификации, результаты которой суммируются в виде конечного продукта.

Модель «Чистой комнаты»

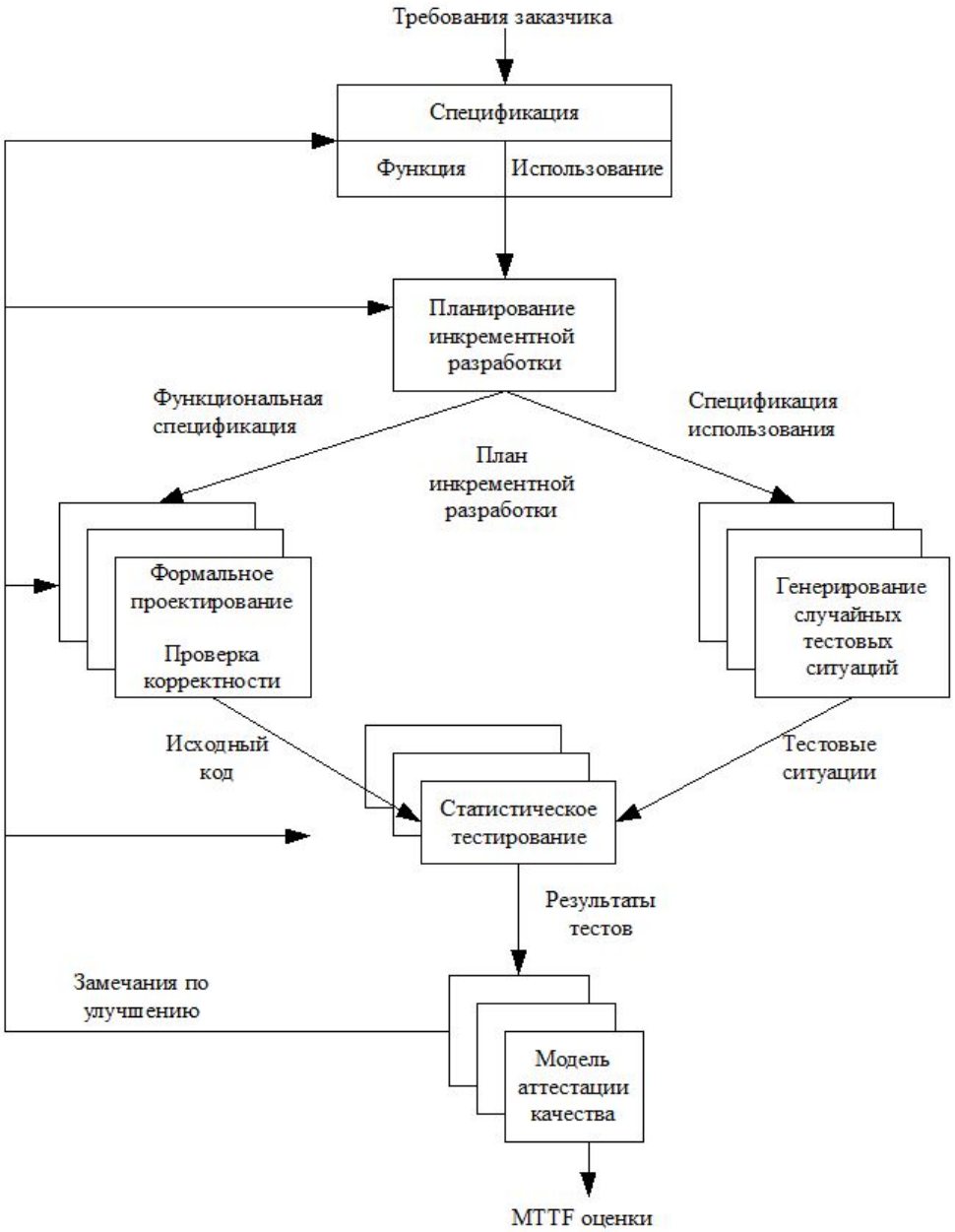
Модель «Чистой комнаты» является одной из моделей формальной разработки систем. Другими моделями данного класса являются Модель разработки Венна (DVM) и нотация Зи (Z notation).

Процессы данной модели приводятся в исполнение малыми, независимыми группами разработки и аттестации ПО. В данной модели всё тестирование основано на предсказывании того, как заказчик будет использовать конечный продукт.

Между данным подходом и каскадной моделью существуют следующие отличия:

1) Спецификация системных требований имеет вид детализированной формальной спецификации, записанной с помощью специальной математической нотации.
2) Процессы проектирования, написания программного кода и тестирования модулей заменяются процессом, в котором формальная спецификация трансформируется в исполняемую программу.

Модель «Чистой комнаты»



Модель «Чистой комнаты»

Достоинства модели:

1) Высокое качество и надежность конечного продукта.
2) Высокая производительность
3) Улучшенная возможность сопровождения
4) Спецификации сильно детализированы и ясны.

Недостатки модели:

1) Данная модель является слишком теоретизированной и требует глубоких знаний в области математики.
2) Большая часть усилий разработчиков уходит на создание спецификаций.
3) Отказ от тестирования отдельных программных модулей.

Модель «Чистой комнаты»

Модель «Чистой комнаты» подходит для очень определённых типов программного обеспечения таких где, возможность появления риска наличия ошибок неприемлема. То есть модель обычно применяется для разработки систем, которые должны удовлетворять строгим требованиям надёжности, безотказности и безопасности.

Модель разработки ПО на основе ранее созданных компонентов

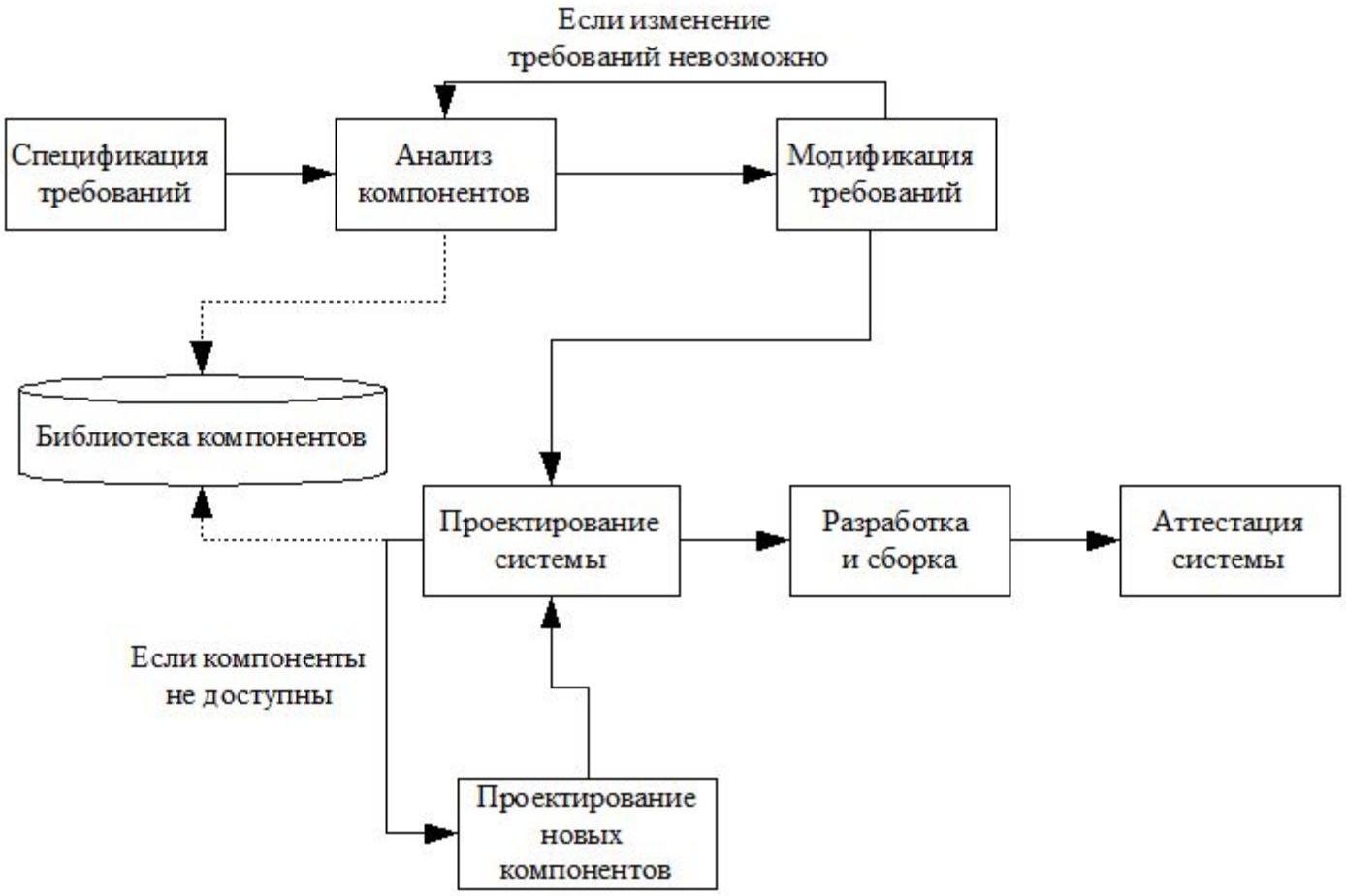
За последние несколько лет данная модель успешно зарекомендовала себя в различных областях разработки ПО. Она применялась в распределённых и web-ориентированных приложениях.

В эволюционной модели для ускорения процесса создания ПО данная модель применяется достаточно часто.

Данная модель основана на наличии большой базы существующих программных компонентов, которые можно интегрировать в создаваемую новую систему.

Подобными компонентами являются свободно продаваемые на рынке программные библиотеки для форматирования текста, числовых вычислений и т.п.

Модель разработки ПО на основе ранее созданных компонентов



Модель разработки ПО на основе ранее созданных компонентов

Достоинства модели:

1) Сокращение количества непосредственно разрабатываемых компонентов.
2) Уменьшается общая стоимость создаваемой системы.
3) Увеличение производительности процесса разработки ПО.

Недостатки модели:

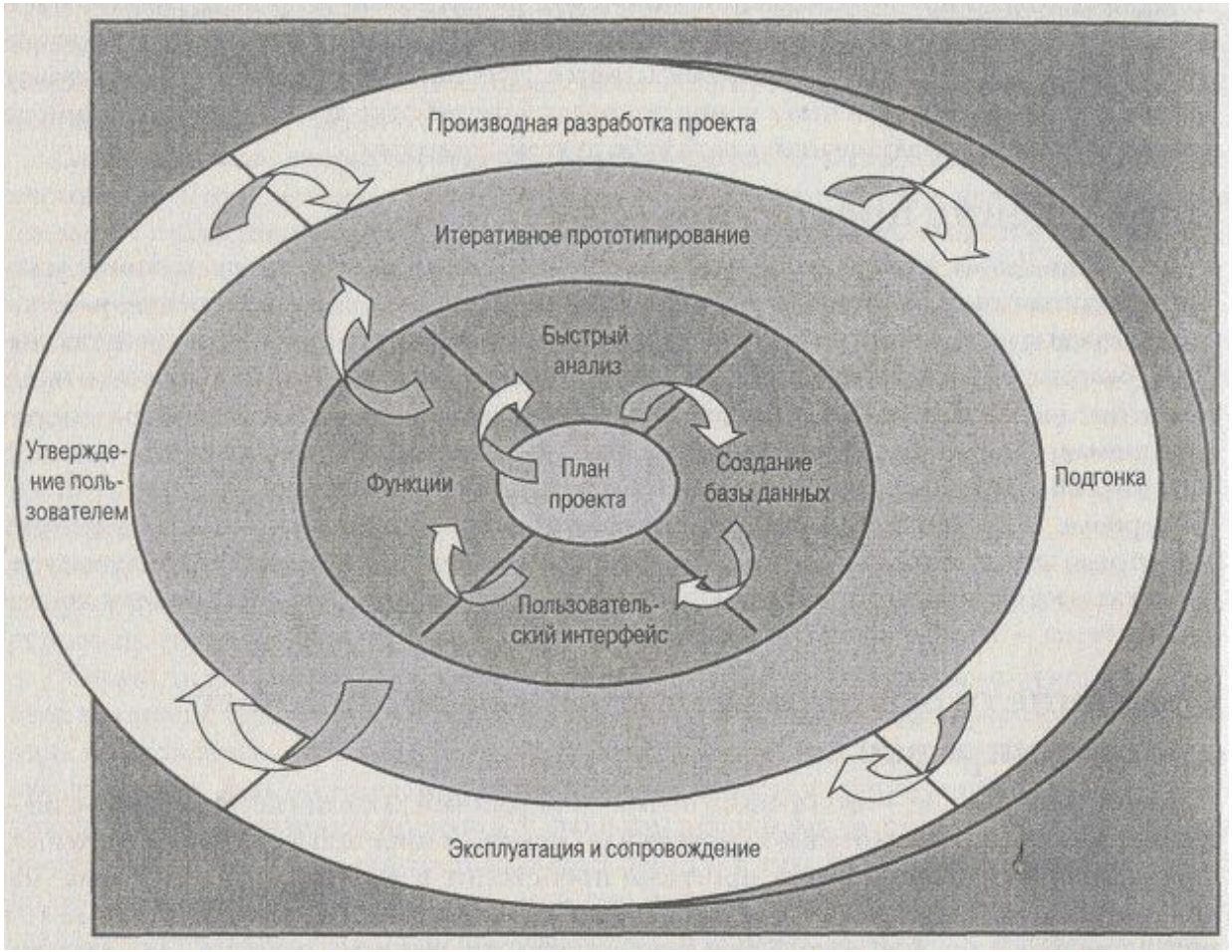
1) Законченная система может не удовлетворять всем требованиям заказчика.
2) При произведении модернизации системы отсутствует возможность влиять на появление новых версий компонентов, используемых в системе.

Модель прототипирования жизненного цикла разработки ПО

Определения прототипирования

- Согласно Джону Коннэллу (Connell) и Линде Шафер (Shafer), эволюционным ускоренным прототипом является "легко поддающаяся модификации и расширению рабочая модель предполагаемой системы, не обязательно представляющая собой все свойства сис темы, благодаря которой пользователи данного приложения получают физическое представление о ключевых частях системы до ее непосредственной реализации; это — легко создаваемая, без труда поддающаяся модификации, максимально расширяемая, частично заданная рабочая модель основных аспектов предполагаемой системы" .
- Бернард Боар (Bernard Boar) определил прототип как "метод, предназначенный для определения требований, при котором потребности пользователя извлекаются, представляются и разрабатываются посредством построения рабочей модели конечной системы — быстро и в требуемом контексте".
- Прототипирование — это процесс построения рабочей модели системы. Прототип — это эквивалент экспериментальной модели или "макета" в мире аппаратного обеспечения.

Структурная эволюционная модель быстрого прототипирования



Преимущества

- конечный пользователь может "увидеть" системные требования в процессе их сбора командой разработчиков; таким образом, взаимодействие заказчика с системой начинается на раннем этапе разработки;
- снижается возможность возникновения путаницы, искажения информации или недоразумений при определении системных требований, что несомненно приводит к созданию более качественного конечного продукта;
- в процесс разработки можно внести новые или неожиданные требования пользователя, что порой необходимо, так как реальность может отличаться от концептуальной модели реальности;
- модель представляет собой формальную спецификацию, воплощенную в рабочую модель;
- модель позволяет выполнять гибкое проектирование и разработку, включая несколько итераций на всех фазах жизненного цикла;
- при использовании модели образуются постоянные, видимые признаки прогресса в выполнении проекта, благодаря чему заказчики чувствуют себя уверенно;
- возможность возникновения разногласий при общении заказчиков с разработчиками минимизирована;
- ожидаемое качество продукта определяется при активном участии пользователя в процесс на ранних фазах разработки;
- благодаря меньшему объему доработок уменьшаются затраты на разработку;
- благодаря тому что проблема выявляется до привлечения дополнительных ресурсов сокращаются общие затраты;
- обеспечивается управление рисками;
- документация сконцентрирована на конечном продукте, а не на его разработке;

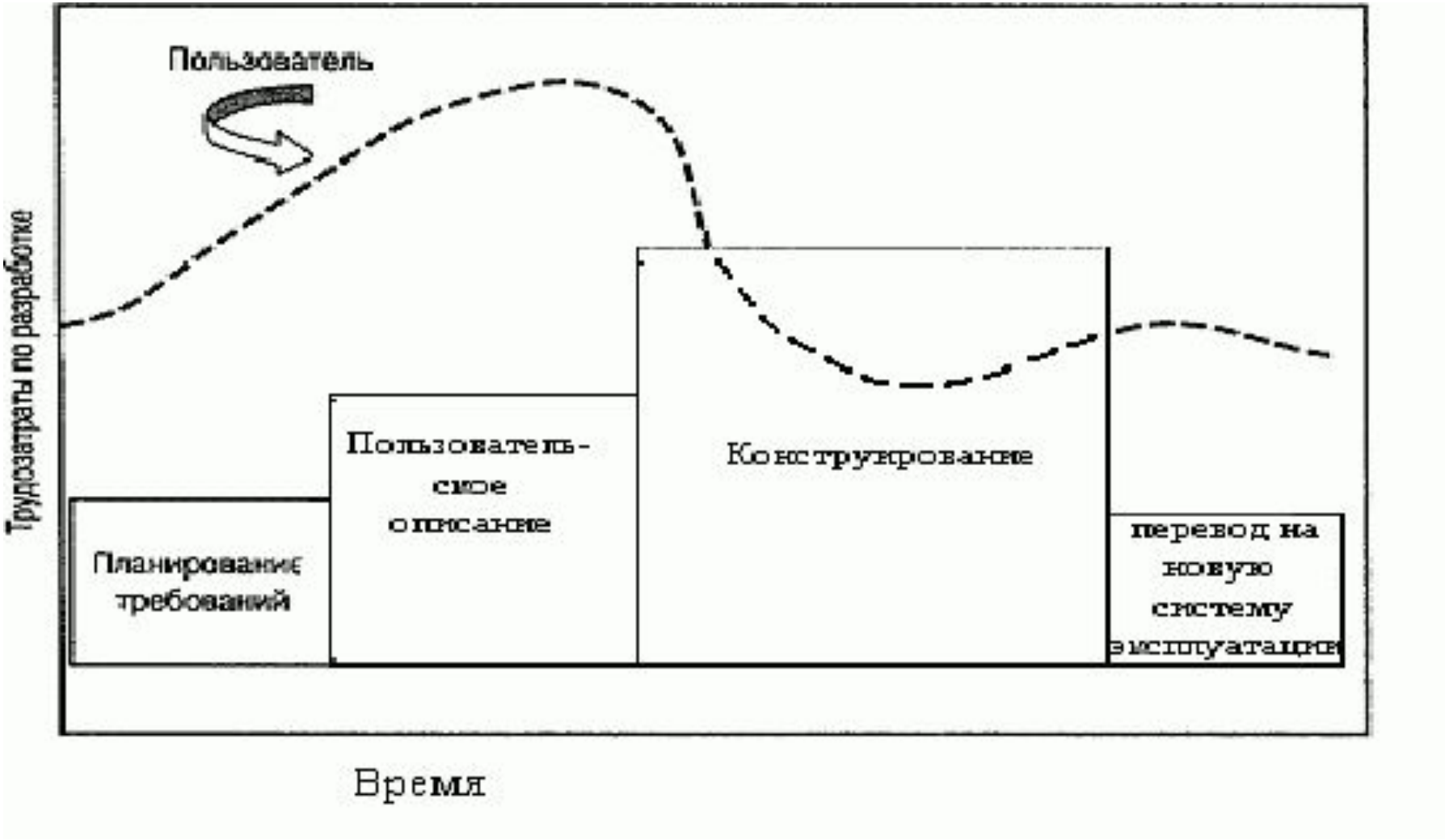
Недостатки

- модель может быть отклонена из-за создавшейся среди консерваторов репутации о ней как о "разработанном на скорую руку" методе;
- разработанные "на скорую руку" прототипы, в отличие от эволюционных ускоренных прототипов, страдают от неадекватной или недостающей документации;
- если цели прототипирования не согласованы заранее, процесс может превратиться в упражнение по созданию хакерского кода;
- с учетом создания рабочего прототипа, качеству всего ПО или долгосрочной эксплуатационной надежности может быть уделено недостаточно внимания.
- иногда в результате использования модели получают систему с низкой рабочей харак теристикой, особенно если в процессе ее выполнения пропускается этап подгонки;
- при использовании модели решение трудных проблем может отодвигаться на бу дущее. В результате это приводит к тому, что последующие полученные продукты могут не оправдать надежды, которые возлагались на прототип;
- если пользователи не могут участвовать в проекте на итерационной фазе быстрого прототипирования жизненного цикла, на конечном продукте могут отразиться неблагоприятные воздействия, включая проблемы, связанные с его качественной характеристикой;
- на итерационном этапе прототипирования быстрый прототип представляет собой частичную систему. Если выполнение проекта завершается досрочно, у конечного пользователя останется только лишь частичная система;
- несовпадение представлений заказчика и разработчиков об использовании прототипа может привести к созданию другого пользовательского интерфейса;

Модель быстрой разработки приложений RAD (Rapid Application Development)

- Благодаря методу RAD пользователь задействован на всех фазах жизненного цикла разработки проекта – не только при определении требований, но и при проектировании, разработке, тестировании, а также конечной поставке про граммного продукта.
- Характерной чертой RAD является короткое время перехода от определения требований до создания полной системы. Метод основывается на последовательности итераций эволюционной системы или прототипов, критический анализ которых обсуждается с заказчиком. В процессе такого анализа формируются требования к продукту.

Модель быстрой разработки приложений



Фазы модели RAD

Модель RAD проходит через следующие фазы:

- **этап планирования требований** — сбор требований выполняется при использовании рабочего метода, называемого совместным планированием требований (Joint requirements planning, JRP), который представляет собой структурный анализ и обсуждение имеющихся коммерческих задач;
- **пользовательское описание** — совместное проектирование приложения (Joint application design, JAD) используется с целью привлечения пользователей; на этой фазе проектирования системы, не являющейся промышленной, работающая над проектом команда зачастую использует автоматические инструментальные средства, обеспечивающие сбор пользовательской информации;
- **фаза конструирования ("до полного завершения") —** эта фаза объединяет в себе детализированное проектирование, построение (кодирование и тестирование), а также поставку программного продукта заказчику за определенное время. Сроки выполнения этой фазы в значительной мере зависит от использования генераторов кода, экранных генераторов и других типов производственных инструментальных средств;
- **перевод на новую систему эксплуатации** — эта фаза включает проведение пользователями приемочных испытаний, установку системы и обучение пользователей.

Преимущества

- время цикла разработки сокращается благодаря использованию мощных инструментальных средств;
- требуется меньшее количество специалистов (поскольку разработка системы выполняется усилиями команды, осведомленной в предметной области);
- существует возможность произвести быстрый изначальный просмотр продукта;
- уменьшаются затраты (благодаря сокращенному времени цикла и усовершенствованной технологии, а также меньшему количеству задействованных в процессе разработчиков);
- благодаря принципу временного блока уменьшаются затраты и риск, связанный с соблюдением графика;
- обеспечивается эффективное использование имеющихся в наличии средств и структур;
- постоянное присутствие заказчика сводит до минимума риск неудовлетворения продуктом и гарантирует соответствие системы коммерческим потребностям и надёжность программного продукта в эксплуатации;
- в состав каждого временного блока входит анализ, проектирование и внедрение (фазы отделены от действий);

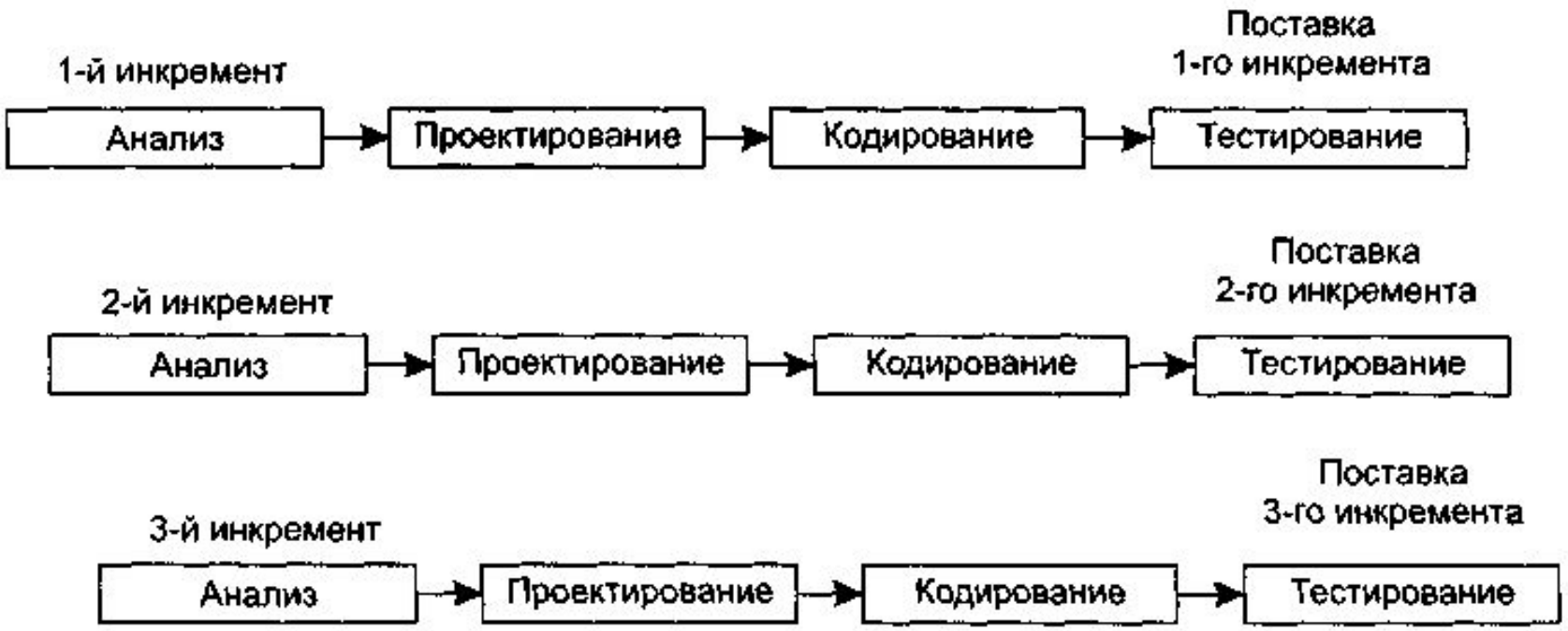
Недостатки

- Непостоянное участие пользователя может негативно сказаться на конечном продукте;
- при использовании этой модели необходимо достаточное количество высоко квалифицированных разработчиков, (умеющих воспользоваться выбранными инструментальными средствами разработки для ускорения времени разработки);
- использование модели может оказаться неудачным в случае отсутствия пригодных для повторного использования компонент;
- могут возникать затруднения при использовании модели совместно с наследственными системами и несколькими интерфейсами;
- возникает потребность в системе, которая может быть смоделирована корректным образом;
- для реализации модели требуются разработчики и заказчики, которые готовы к быстрому выполнению действий ввиду жестких временных ограничений;
- для обеспечения быстрой реакции на информацию, поступающую в результате налаженной обратной связи с пользователем, необходим эффективный ускоренный процесс разработки.
- при использовании модели "вслепую" на затраты и дату завершения работы над проектом ограничения не накладываются;

Инкрементная модель жизненного цикла разработки ПО

- Инкрементная разработка представляет собой процесс частичной реализации всей системы и медленного наращивания функциональных возможностей. Этот подход позволяет уменьшить затраты, понесенные до момента достижения уровня исходной производительности.
- Инкрементная модель действует по принципу каскадной модели с перекрытиями, благодаря чему функциональные возможности продукта, пригодные к эксплуатации, формируется раньше.

Инкрементная модель



Преимущества

- не требуется заранее тратить средства, необходимые для разработки всего проекта (поскольку сначала выполняется разработка и реализация основной функции или функции из группы высокого риска);
- в результате выполнения каждого инкремента получается функциональный продукт;
- заказчик располагает возможностью высказаться по поводу каждой разработанной версии системы;
- правило по принципу "разделяй и властвуй" позволяет разбить возникшую про блему на управляемые части, благодаря чему предотвращается формирование громоздких перечней требований, выдвигаемых перед командой разработчиков;
- существует возможность поддерживать постоянный прогресс в ходе выполнения проекта;
- снижаются затраты на первоначальную поставку программного продукта;
- ускоряется начальный график поставки (что позволяет соответствовать возрос шим требованиям рынка);
- снижается риск неудачи и изменения требований;
- заказчики могут распознавать самые важные и полезные функциональные воз можности продукта на более ранних этапах разработки;
- риск распределяется на несколько меньших по размеру инкрементов (не сосре доточен в одном большом проекте разработки);

Недостатки

- в модели не предусмотрены итерации в рамках каждого инкремента;
- определение полной функциональной системы должно осуществляться в начале жизненного цикла, чтобы обеспечить определение инкрементов;
- формальный критический анализ и проверку намного труднее выполнить для инкрементов, чем для системы в целом;
- заказчик должен осознавать, что общие затраты на выполнение проекта не будут снижены;
- поскольку создание некоторых модулей будет завершено значительно раньше других, возникает необходимость в четко определенных интерфейсах;
- использование на этапе анализа общих целей, вместо полностью сформулированных требований, может оказаться неудобным для руководства;
- для модели необходимы хорошее планирование и проектирование: руководство должно заботиться о распределении работы, а технический персонал должен соблюдать субординацию в отношениях между сотрудниками.
- может возникнуть тенденция к оттягиванию решений трудных проблем на будущее с целью продемонстрировать руководству успех, достигнутый на ранних этапах разработки;

Список литературы

1) S. Kan. Metrics and Models in Software Quality Engineering. Addison Wesley, 2002.
2) И. Соммервилл. Инженерия программного обеспечения. Издательский дом «Вильямс», 2002.
3) С. Орлов. Технологии разработки программного обеспечения. СПб:Питер, 2002.
4) В. Липаев. Программная инженерия. Издательство «ТЕИС», 2006.
5) E. May, B. Zimmer. The Evolutionary Development Model for Software // Hewlett-Packard Journal, August 1996.
6) R. Linger. Cleanroom Software Engineering for Zero Defect Software // IEEE, 1993.