ПМЗ
МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ПО

Лекция 4-1. Классификация технологических подходов (модели жизненного цикла): **гибкие подходы**

Технологические подходы

1. Подходы со слабой формализацией

2 Строгие (классические, жесткие, предсказуемые) подходы

- 2.1. Каскадные технологические подходы.
  - 2.1.1. Классический каскадный подход
  - 2.1.2. Каскадно-возвратный подход
  - 2.1.3. Каскадно-итерационный подход
  - 2.1.4. Каскадный подход с перекрывающимися процессами
  - 2.1.5. Каскадный подход с подпроцессами
  - 2.1.6. Спиральная модель
- 2.2. Каркасные подходы.
  - 2.2.1. Рациональный унифицированный процесс (RUP)
- 2.3. Генетические подходы.
  - 2.3.1. Синтезирующее программирование
  - 2.3.2. Сборочное (расширяемое) программирование
  - 2.3.3. Конкретизирующее программирование
- 2.4. Подходы на основе формальных преобразований.
  - 2.4.1. Технология стерильного цеха
  - 2.4.2. Формальные генетические подходы

3 Гибкие (адаптивные, легкие) подходы

- 3.1. Ранние технологические подходы быстрой разработки (RAD)
  - 3.1.1. Эволюционное прототипирование
  - 3.1.2.Итеративная разработка
  - 3.1.3. Постадийная разработка
- 3.2. Адаптивные подходы.
  - 3.2.1. Экстремальное программирование (XP)
  - 3.2.2. Адаптивная разработка
- 3.3. Подходы исследовательского программирования.
  - 3.3.1. Компьютерный дарвинизм

3. Гибкие (адаптивные, легкие) подходы

подходов быстрой разработки (RAD)

Развитием и одновременно альтернативой каскадных подходов является группа подходов быстрой разработки. Все эти подходы объединяют следующие основные черты:

- итерационную разработку прототипа;
- тесное взаимодействие с заказчиком.

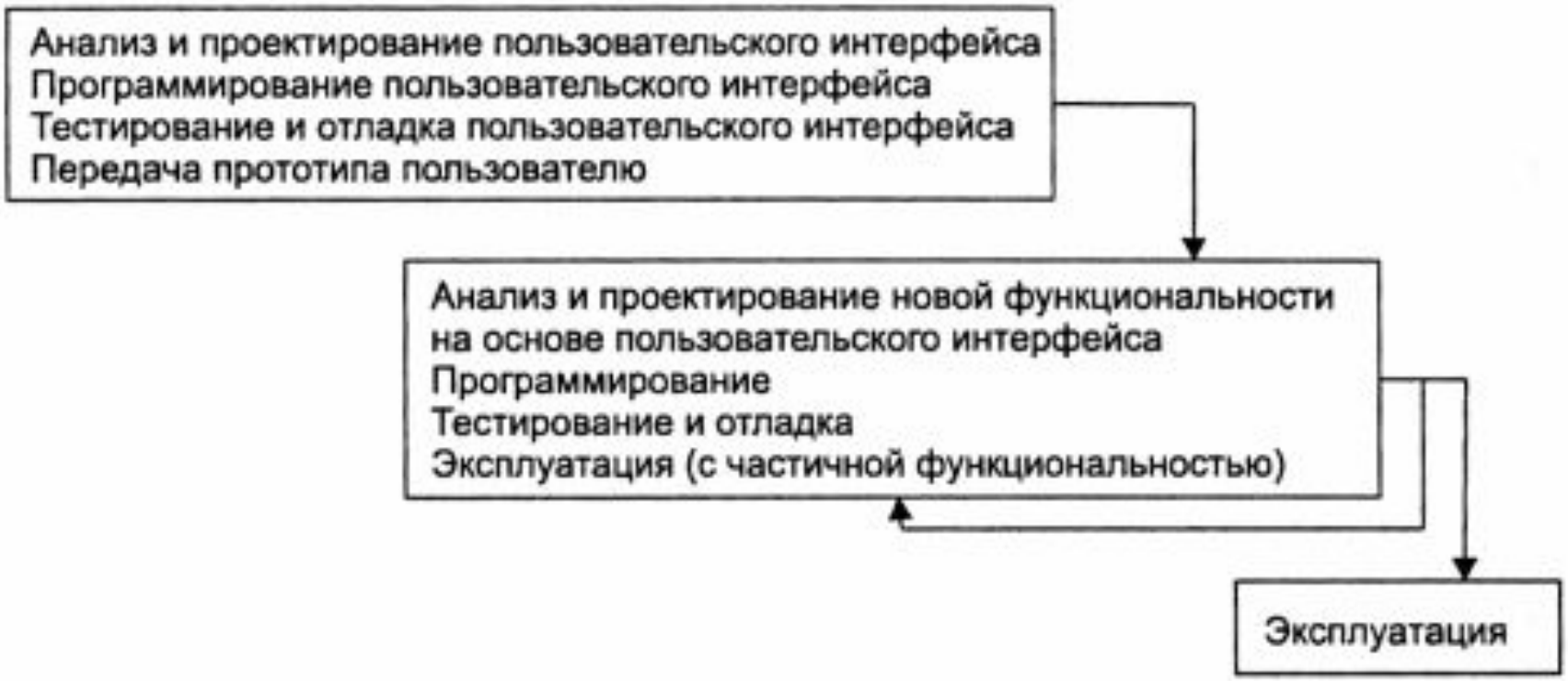
3.1.1. Эволюционное прототипирование (макетирование)

Первый прототип обычно включает создание развитого **пользовательского интерфейса**. Он может быть сразу же продемонстрирован заказчику для получения от него отзывов и возможных корректив. Основное начальное внимание уделяется стороне системы, обращенной к пользователю.

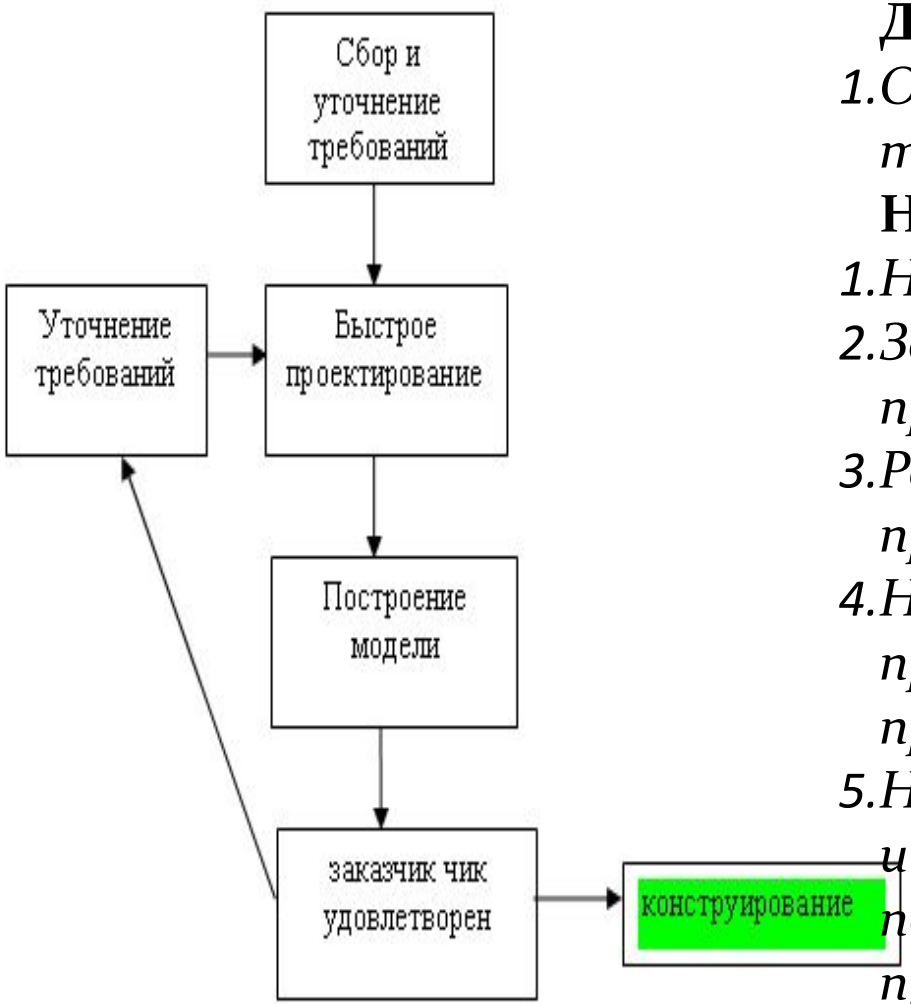
Разумно применять в тех случаях, когда заказчик не может четко сформулировать свои требования к программному продукту на начальных этапах разработки или заказчик знает, что требования могут кардинально измениться.

3.1.1. Эволюционное прототипирование (макетирование)

Далее, до тех пор, пока пользователь не сочтет программный продукт законченным, в него вносится необходимая функциональность.



3.1.1. Эволюционное прототипирование (макетирование)



Достоинства:

1. *Обеспечивает определение полных требований*

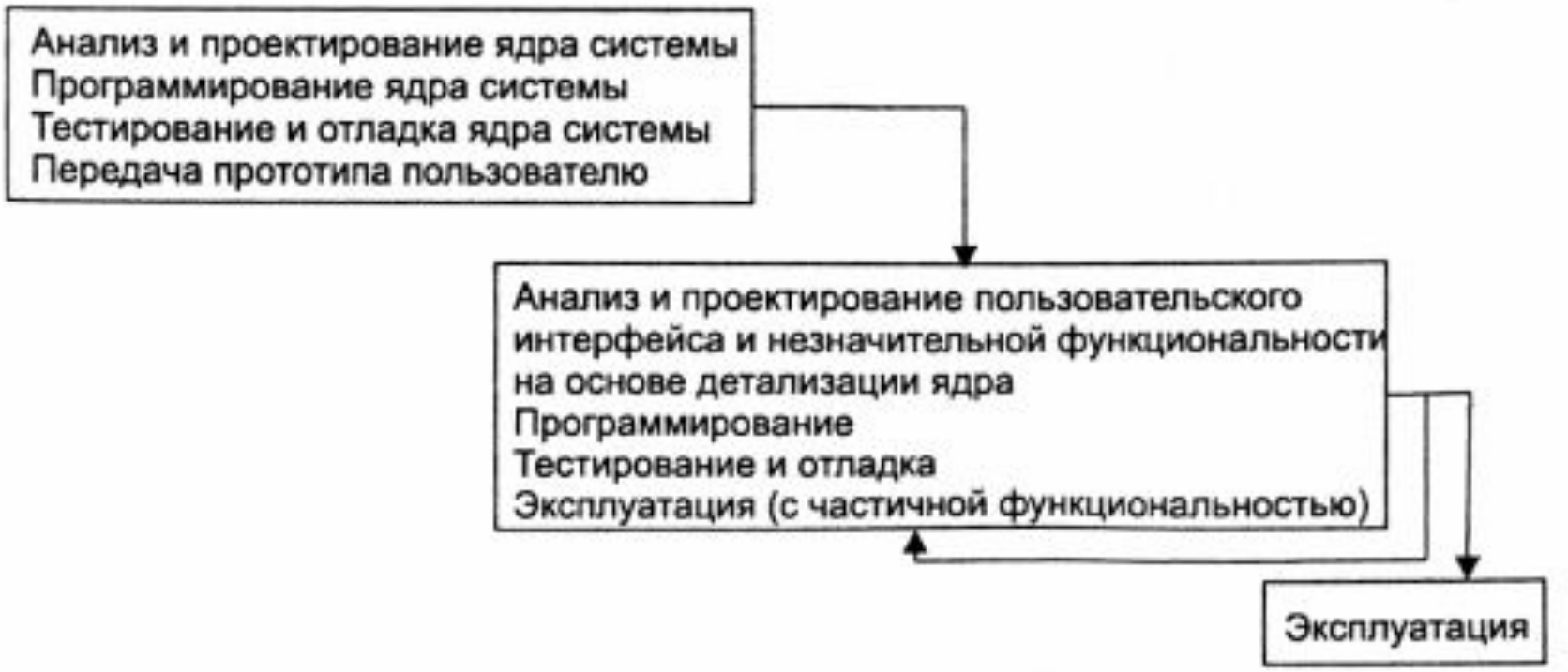
Недостатки:

1. *Не является полным ЖЦ*
2. *Заказчик может принять макет за продукт*
3. *Разработчик может принять макет за продукт*
4. *Невозможность определить продолжительность и стоимость проекта*
5. *Неочевидным является количество итераций, по истечении которых пользователь сочтет программный продукт законченным.*

3.1.2. Итеративная разработка

Первый прототип итеративной разработки (iterative delivery) уже должен включать завершенное **ядро системы**. Таким образом, в нем уже сосредоточена **большая часть функциональности**. Очередные итерации должны помочь пользователю определиться с доводкой пользовательского интерфейса, генерируемых системой отчетов и других выходных данных.

разработка



Допускается добавление незначительной функциональности, обычно не затрагивающей ядро системы.

3.1.2. Различие между ЭВОЛЮЦИОННЫМ и ИТЕРАТИВНЫМ методами быстрой разработки

- ЭВОЛЮЦИОННОЕ прототипирование ориентировано на начальную разработку пользовательского интерфейса и добавления функциональности на его основе.
- ИТЕРАТИВНАЯ разработка начинается с создания ядра системы и далее детализирует его.

(инкрементальная) разработка

Постадийная разработка (staged delivery) предназначена решить недостаток двух предыдущих подходов - **невозможность определения сроков завершения** проекта.

Изначально достаточно хорошо известно, что будет собой представлять создаваемый программный продукт. Основная задача постадийной разработки - предоставить заказчику работающую систему как можно **раньше**.

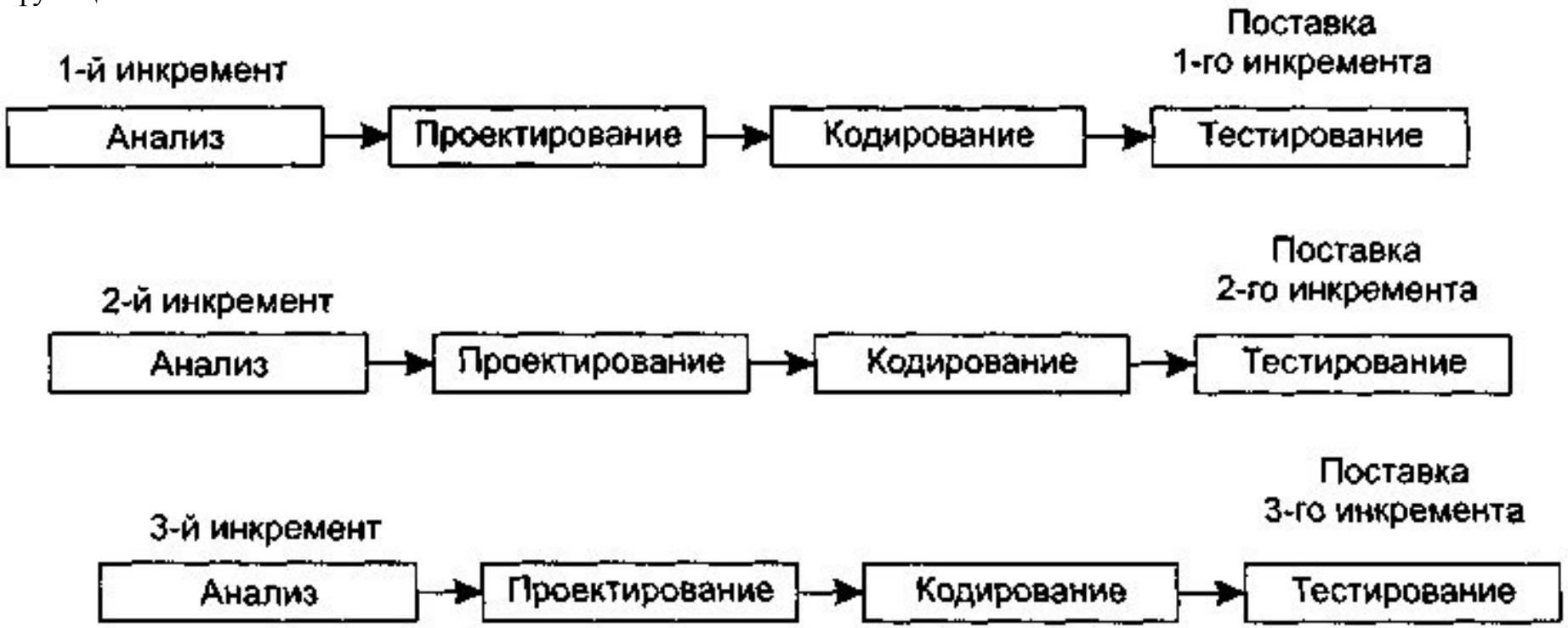
Далее заказчик сможет добавлять новую функциональность и получать очередную версию системы.

Каждая из версий, получаемых по завершении стадий, является работающей.

Подход требует очень тщательного и серьезного тестирования очередной системы в конце каждой стадии перед передачей ее пользователю.

(инкрементальная) разработка

Инкрементная модель объединяет элементы последовательной водопадной модели и итерационную философию макетирования. Каждая линейная последовательность здесь вырабатывает поставляемый инкремент ПО. Например, в ПО обработки слов 1-й инкремент реализует функции базовой обработки файлов (редактирования и документирования); 2-й инкременте – проверку орфографии и грамматики; 3-й инкремент – возможности компоновки страницы. Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными). План следующего инкремента предусматривает модификацию базового продукта, обеспечивающую дополнительные характеристики и функциональность.



Технологические подходы

1. Подходы со слабой формализацией

2 Строгие (классические, жесткие, предсказуемые) подходы

- 2.1. Каскадные технологические подходы.
  - 2.1.1. Классический каскадный подход
  - 2.1.2. Каскадно-возвратный подход
  - 2.1.3. Каскадно-итерационный подход
  - 2.1.4. Каскадный подход с перекрывающимися процессами
  - 2.1.5. Каскадный подход с подпроцессами
  - 2.1.6. Спиральная модель
- 2.2. Каркасные подходы.
  - 2.2.1. Рациональный унифицированный процесс (RUP)
- 2.3. Генетические подходы.
  - 2.3.1. Синтезирующее программирование
  - 2.3.2. Сборочное (расширяемое) программирование
  - 2.3.3. Конкретизирующее программирование
- 2.4. Подходы на основе формальных преобразований.
  - 2.4.1. Технология стерильного цеха
  - 2.4.2. Формальные генетические подходы

3 Гибкие (адаптивные, легкие) подходы

- 3.1. Ранние технологические подходы быстрой разработки (RAD)
  - 3.1.1. Эволюционное прототипирование
  - 3.1.2.Итеративная разработка
  - 3.1.3. Постадийная разработка
- 3.2. Адаптивные подходы.
  - 3.2.1. Экстремальное программирование (XP)
  - 3.2.2. Адаптивная разработка
- 3.3. Подходы исследовательского программирования.
  - 3.3.1. Компьютерный дарвинизм

Статистика использования методологий

В 2009г. опросили более 1000 различных ИТ разработчиков (какую методологию проектирования они используют?)
Результат опроса (по количеству проектов):

35% - гибкие (Scrum около 10%, XP около 2,5% и др)
30% - не используют никаких
21% - разные варианты итеративных (эволюционных) процессов (RUP, спиральные и др.)
13% - различные модификации классической (водопадной) модели

3.2. Адаптивные технологические подходы

Адаптивные технологические подходы были задуманы как подходы, поддерживающие изменения. Они только выигрывают от изменений, даже когда изменения происходят в них самих. Данные подходы ориентированы на человека, а не на процесс. В них необходимо учитывать в работе природные качества человеческой натуры, а не действовать им наперекор.

Все гибкие методологии разрабатываются параллельно

Гибкие (agile) методологии Основные особенности

- Отказ от классических неповоротливых подходов
- Направленность на проекты с постоянно меняющимися требованиями
- Небольшие команды
- (!) Высокая значимость не только технических составляющих процесса, но и организационных, социальных и т.п.

Манифест гибкой разработки ПО (Agile Manifesto), 2001 год

- Люди и взаимодействие
- Работающий продукт
- Сотрудничество с заказчиком
- Готовность к изменениям

важнее

- Процессов и инструментов
- Исчерпывающей документации
- Согласования условий контракта
- Следования первоначальному плану

1) Процессы и инструменты важны при разработке программного обеспечения. Однако успех проекта в гораздо большей степени зависит от людей, участвующих в нём, и их способности эффективно общаться и взаимодействовать. Речь идёт о главенствующей роли человеческого фактора в разработке ПО.
2) Подчёркивается, что целью и мерилом успеха проекта является не документация, а работоспособное программное обеспечение, решающее бизнес-задачи заказчика. Документация нужна не сама по себе, а лишь в том минимальном объёме, который необходим для создания качественного программного продукта.
3) Доверие между командой и заказчиком, а также непрерывное общение и сотрудничество являются гораздо более выигрышной стратегией, чем попытка опираться исключительно на формальные договорённости, такие, как контракт (что не отменяет необходимости создания и проработки контракта).
4) Изменения (прежде всего, в требованиях) являются не только неизбежным, но и необходимым элементом практически любого проекта. Проекты делаются для того, чтобы приносить пользу и ценность бизнесу заказчика. Эти цели могут быть достигнуты, даже если план выполнен не идеально с точки зрения изначально определённых сроков и бюджета. И наоборот, может случиться так, что выполненный точно в срок и в соответствии с исходными требованиями проект не принесет никакой ценности заказчику (например, из-за того, что в ходе проекта изменился бизнес заказчика или внешняя среда, и исходные требования перестали быть актуальными). Гибкие методы ни в коей мере не подвергают сомнению необходимость планирования - они лишь признают его объективные ограничения в условиях высокой неопределённости и быстро меняющейся внешней среды.

3.2.1. Экстремальное программирование (XP)

Наиболее концентрированно идеи быстрой разработки программ оказались выражены в подходе экстремального программирования (extreme programming). Две основные черты, присущие быстрым разработкам, являются базовыми и в этом подходе. Методы, объединенные в данном подходе, не являются принципиально новыми. Однако именно их рациональное объединение и совокупное использование дает существенные результаты и успешно выполненные проекты.

Наибольшую пользу подход экстремального программирования может принести в разработке

- **небольших систем**,
- требования к которым
  - четко не определены
  - могут измениться.

3.2.1. Экстремальное программирование (XP)

Методология XP была создана Кентом Беком (Kent Beck) в 1996 году в ходе попытки спасти провальный проект по разработке системы расчета зарплаты для компании Крайслер. В 2000 году проект был закрыт, но XP к тому времени уже получила известность и начала распространяться среди разработчиков ПО. XP наследует все общие принципы гибких методологий, достигая их при помощи ДВЕНАДЦАТИ ПРИНЦИПОВ :

1. **Планирование процесса**. Вся команда разработчиков собирается вместе, принимается коллективное решение о том, какие свойства системы будут реализованы в ближайшей итерации. Трудоемкость реализации каждого свойства определяется самими программистами.
2. **Тесное взаимодействие с заказчиком**. Представитель заказчика должен быть членом XP-команды. Он пишет ПИ, выбирает истории, которые будут реализованы в конкретной итерации, и отвечает на вопросы, касающиеся бизнеса. Представитель заказчика должен быть экспертом в автоматизируемой предметной области. Необходимо наличие постоянное обратной связи с представителем заказчика.
3. **Общесистемные правила именования**. Хорошие системные правила именования предполагают **простоту именования** классов и переменных. Команда разработчиков должна иметь единые правила именования.

4. **Простая архитектура**. Любое свойство системы должно быть реализовано как можно проще. Программисты в XP-команде работают под девизом: «Ничего лишнего» . Принимается первое простейшее работающее решение, реализуется необходимый уровень функциональности на данный момент. Тем самым экономится время программиста.
5. **Рефакторинг**. Это оптимизация существующего кода с целью его упрощения, Такая работа должна вестись постоянно. Сохраняя код прозрачным и определяя его элементы всего один раз, программисты сокращают число ошибок, которые впоследствии придется устранять. При реализации каждого нового свойства системы программист должен подумать над тем, можно ли упростить существующий код и как это поможет реализовать новое свойство. Кроме того, нельзя совмещать рефакторинг с дизайном: если создается новый код, рефакторинг следует отложить.
6. **Парное программирование**. Все программисты должны работать в парах: один пишет код, другой смотрит. Таким образом, необходимо размещать группу программистов в одном месте. XP наиболее успешно работает в нераспределенных коллективах программистов и пользователей.
7. **40-часовая рабочая неделя**. Программист не должен работать более 8 часов в день. Необходимость сверхурочной работы - это четкий индикатор проблемы на данном конкретном направлении разработки. Поиск причин сверхурочной работы и их скорейшее устранение - одно из основных правил.

8. **Коллективное владение кодом**. Каждый программист в коллективе должен иметь доступ к коду любой части системы и право вносить изменения в любой код. Обязательное правило: если программист внес изменения и система после этого работает некорректно, то именно этот программист должен исправить ошибки.
9. **Единые стандарты кодирования**. Стандарты кодирования нужны для обеспечения других практик: коллективного владения кодом, парного программирования и рефакторинга. Без единого стандарта выполнять эти практики как минимум сложнее, а в реальности вообще невозможно: группа будет работать в режиме постоянной нехватки времени. Команда работает над проектом продолжительное время. Люди приходят и уходят. Никто не кодирует в одиночку и код принадлежит всем. Всегда будут моменты, когда необходимо будет понять и скорректировать чужой код. Разработчики будут удалять дублирующий код, анализировать и улучшать чужие классы и т. п. Со временем нельзя будет сказать, кто автор конкретного класса. Следовательно, все должны подчиняться общим стандартам кодирования - форматирование кода, именование классов, переменных, констант, стиль комментариев. Вышесказанное означает, что все члены команды должны договориться об общих стандартах кодирования. Неважно каких, но все обязаны им подчиняются.

0. **Небольшие релизы (**версии - releases) . Минимальная итерация - один день, максимальная - месяц; чем чаще осуществляются релизы, тем больше недостатков системы будет выявлено. Первые релизы помогают выявить недостатки на самых ранних стадиях, далее функциональность системы расширяется на основании пользовательских историй (ПИ). Пользователь включается в процесс разработки с первого релиза, может оценить систему, выдать ПИ и замечания. На основании этого определяется следующая итерация, т.е. каким будет новый релиз. В XP все направлено на обеспечение непрерывной обратной связи с пользователями.
1. **Непрерывная интеграция**. Интеграция новых частей системы должна происходить как можно чаще, как минимум раз в несколько часов. Основное правило интеграции следующее: интеграцию можно производить, если все тесты проходят успешно. Если тесты не проходят, то программист должен либо внести исправления и тогда интегрировать составные части системы, либо вообще не интегрировать их. Правило это - жесткое и однозначное. Если в созданной части системы имеется хотя бы одна ошибка, то интеграцию производить нельзя. Частая интеграция позволяет быстрее получить готовую систему, вместо того чтобы тратить на сборку неделю.
2. **Тестирование**. В отличие от большинства остальных методологий тестирование в XP - одно из важнейших составляющих. Тесты пишутся до написания кода. Каждый модуль обязан иметь unit test (тест данного модуля). Таким образом, в XP осуществляется регрессионное тестирование, при добавлении функциональности. Большинство ошибок исправляются на стадии кодирования. Тесты пишут сами программисты, любой из них имеет право написать тест для любого модуля. Еще один важный принцип: тест определяет код, а не наоборот (test-driven development), то есть кусок кода кладется в хранилище тогда и только тогда, когда все тесты прошли успешно, в противном случае данное изменение кода отвергается.

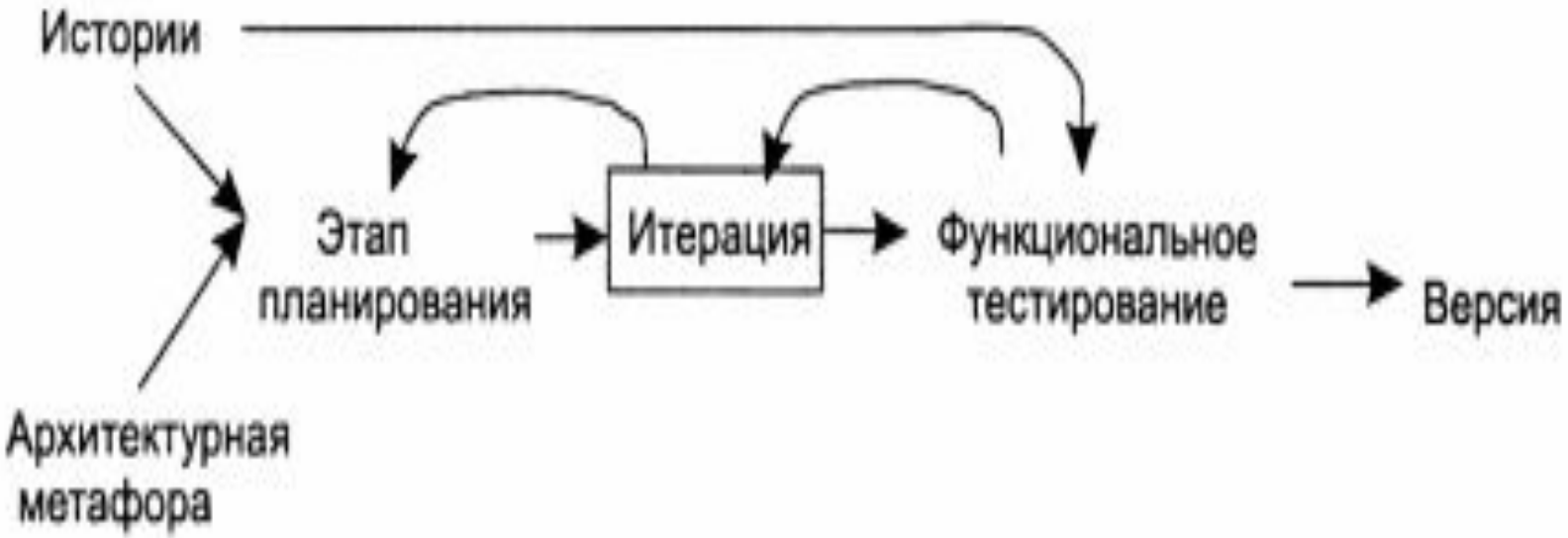
3.2.1. Экстремальное программирование

Подход начинается с того, что проводится анализ назначения системы и определения первоочередной функциональности. В результате составляется список историй - возможных применений системы. Каждая история должна быть ориентирована на определенные задачи бизнеса, которые можно оценить с помощью количественных показателей. Наконец, ценность истории определяется материальными и временными затратами на ее разработку командой разработчиков.

Заказчик выбирает истории для очередной итерации, основываясь на их значимости для проекта и ценности. Для первой версии системы заказчик определяет небольшое количество логически связанных наиболее важных историй. Для каждой следующей версии выбираются наиболее важные истории из числа оставшихся.

3.2.1. Экстремальное программирование

Работа над проектом на основе экстремального программирования



Пользовательские истории – короткие неформальные описания прецедентов использования системы. В XP истории являются основным и единственным средством спецификации требований и приемочными тестами.

Архитектурная метафора даёт команде представление о том, каким образом система работает в настоящее время, в каких местах добавляются новые компоненты, и какую форму они должны принять. Подбор хорошей метафоры облегчает для группы разработчиков понимание того, каким образом устроена система. (Архитектура — это представление о компонентах системы и их взаимосвязях между собой.)

3.2.1. Экстремальное программирование

Одним из существенных методов данного подхода является функциональное тестирование. Существуют две особенности процесса тестирования.

- Программисты сами пишут тесты для тестирования программы.
- Эти тесты пишутся до начала кодирования.

Уверенность в нормальной работе каждого отдельного теста постепенно формирует у разработчиков уверенность в нормальной работе системы в целом.

Цель каждой итерации - включить в версию несколько новых историй. На собрании по планированию итерации определяется, какие именно истории будут реализованы и каким образом это будет сделано командой разработчиков.

3.2.1. Экстремальное программирование

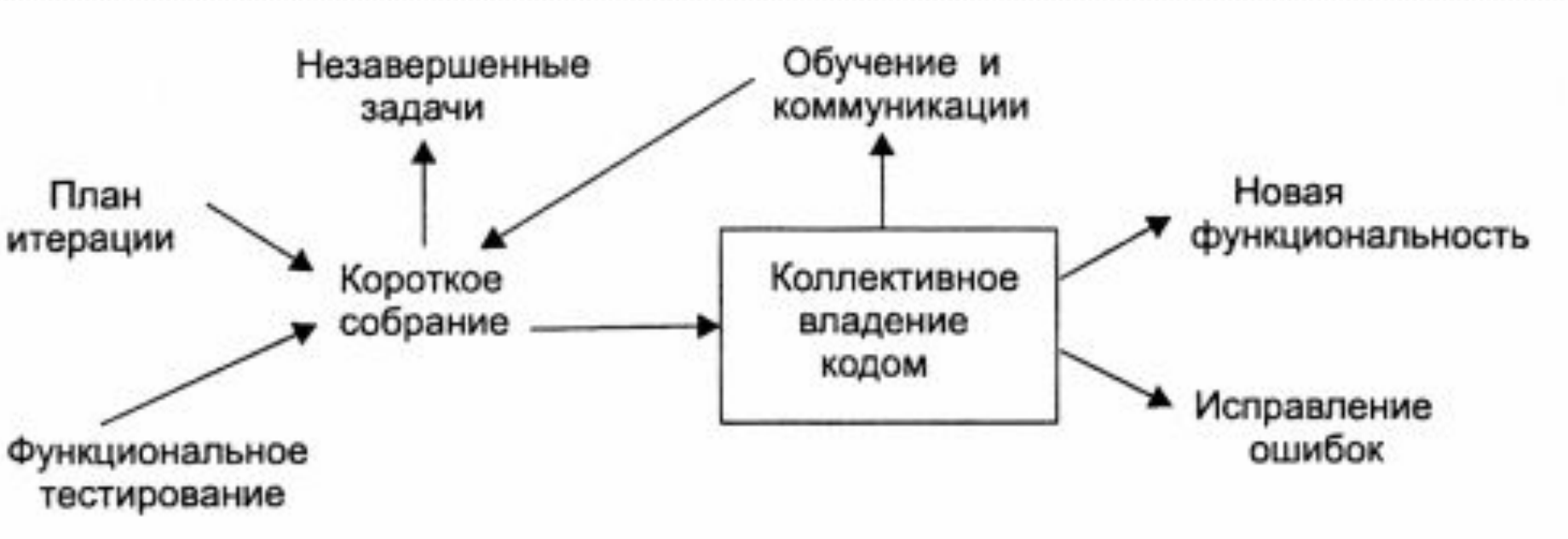
- Итерации в схеме XP:

3.2.1. Экстремальное программирование

- Коллективное владение кодом в процессе разработки означает возможность для каждого программиста в любое время усовершенствовать любую часть кода в системе, если он сочтет это необходимым. Программист берет на себя ответственность за реализацию определенных задач. В случае возникновения вопросов о разрабатываемой задаче, может быть проведено короткое (15-минутное) собрание с присутствием заказчика.

3.2.1. Экстремальное программирование

- Разработка (в схеме итерации XP) :



3.2.1. Экстремальное программирование

- Для того чтобы выполнить задачу, ответственный за нее программист должен найти себе партнера. Окончательный код всегда пишется двумя программистами на одной рабочей станции.
- Коллективное владение кодом:

3.2.1. Экстремальное программирование

- Экстремальное программирование уделяет значительное внимание организации офисного пространства, отмечая существенное влияние окружающих условий на работу программистов

3.2.1. Экстремальное программирование. Выводы

Методология XP является сегодня одной из наиболее известных среди гибких методологий. Иногда само понятие «гибкие методологии» явно или неявно отождествляют с XP, которая проповедует коммуникабельность, простоту, обратную связь и отвагу. Она описывается как набор практик: игра в планирование, короткие релизы, метафоры, простой дизайн, переработка кода (refactoring), разработка «тестами вперед», парное программирование, коллективное владение кодом, 40-часовая рабочая неделя, постоянное присутствие заказчика и стандарты кода. Интерес к XP рос снизу вверх — от разработчиков и тестировщиков, замученных тягостным процессом, документацией, метриками и прочим формализмом. Они не отрицали дисциплину, но не желали бессмысленно соблюдать формальные требования и искали новые быстрые и гибкие подходы к разработке высококачественных программ.

3.2.1. Экстремальное программирование. Выводы

При использовании XP тщательное предварительное проектирование ПО заменяется, с одной стороны, постоянным присутствием в команде заказчика, готового ответить на любой вопрос и оценить любой прототип, а с другой — регулярными переработками кода (так называемый рефакторинг). Основой проектной документации считается тщательно прокомментированный код. Очень большое внимание в методологии уделяется тестированию. Как правило, для каждого нового метода сначала пишется тест, а потом уже разрабатывается собственно код метода до тех пор, пока тест не начнет выполняться успешно. Эти тесты сохраняются в наборах, которые автоматически выполняются после любого изменения кода.

Хотя парное программирование и 40-часовая рабочая неделя и являются, возможно, наиболее известными чертами XP, но все же носят **вспомогательный характер** и способствуют высокой производительности разработчиков и сокращению количества ошибок при разработке.

3.2.2. Адаптивная разработка

- В основу подхода адаптивной разработки (Adaptive Software Development - ASD) положены три нелинейные перекрывающие друг друга фазы - обдумывание, сотрудничество и обучение. Автор данного подхода Джим Хайсмит (Jim Highsmith) обращает особое внимание на использование идей из области сложных адаптивных систем.
- Результаты планирования (которое само здесь является парадоксальным) в адаптивной разработке всегда будут непредсказуемыми. При обычном планировании отклонение от плана является ошибкой, которую исправляют. При данном подходе отклонения ведут к правильным решениям.
- Программисты должны активно сотрудничать между собой для преодоления неопределенности в подходе адаптивной разработки. Руководители проектов должны обеспечить хорошие коммуникации между программистами. Благодаря этому программисты сами находят объяснения на возникающие вопросы, а не ждут их от руководителей.
- Обучение является постоянной и важной характеристикой подхода. И программисты, и заказчики в процессе работы должны пересматривать собственные обязательства и планы. Итоги каждого цикла разработки используются при подготовке следующего.

3.2.2. Scrum

- **Scrum** (от англ. *scrum* «толкучка») — методология управления проектами, активно применяющаяся при разработке информационных систем для гибкой разработки ПО.
- Scrum чётко делает акцент на качественном контроле процесса разработки (управлении проектами по разработке ПО).
- Scrum может также использоваться в работе команд поддержки ПО (*software support teams*), или как подход управления разработкой и сопровождением программ: *Scrum of Scrums*.

SCRUM

- Начиная с 1986 г. по 2001г. ряд авторов (Такэути, Нонака, Швабер, Сазерленд и Бидл) документировали, оформляли и описывали подход, детально описанный, наконец, как отдельный метод в книге «Agile Software Development with SCRUM».

Они отмечали, что проекты, над которыми работают **НЕБОЛЬШИЕ** команды из специалистов различного профиля, обычно систематически производят лучшие результаты, и объяснили это как «подход регби».

SCRUM. Артефакты

- **Backlog (Бэклог проекта)**
  - Список работ, которые необходимо выполнить
- **Sprint backlog**
  - Набор требований, которые могут быть реализованы за один этап (спринт)

SCRUM. Планирование

- Спринт (**Sprint)**

30-дневный (обычно) промежуток, за который выполняется реализация заданной функциональности

Планирование происходит между спринтами. Команда самоорганизуется

- Планирование спринта

Происходит в начале спринта (исходя из приоритета и необходимой функциональности)

- Scrum

Ежедневная встреча разработчиков (каждый день, 10-15 мин)

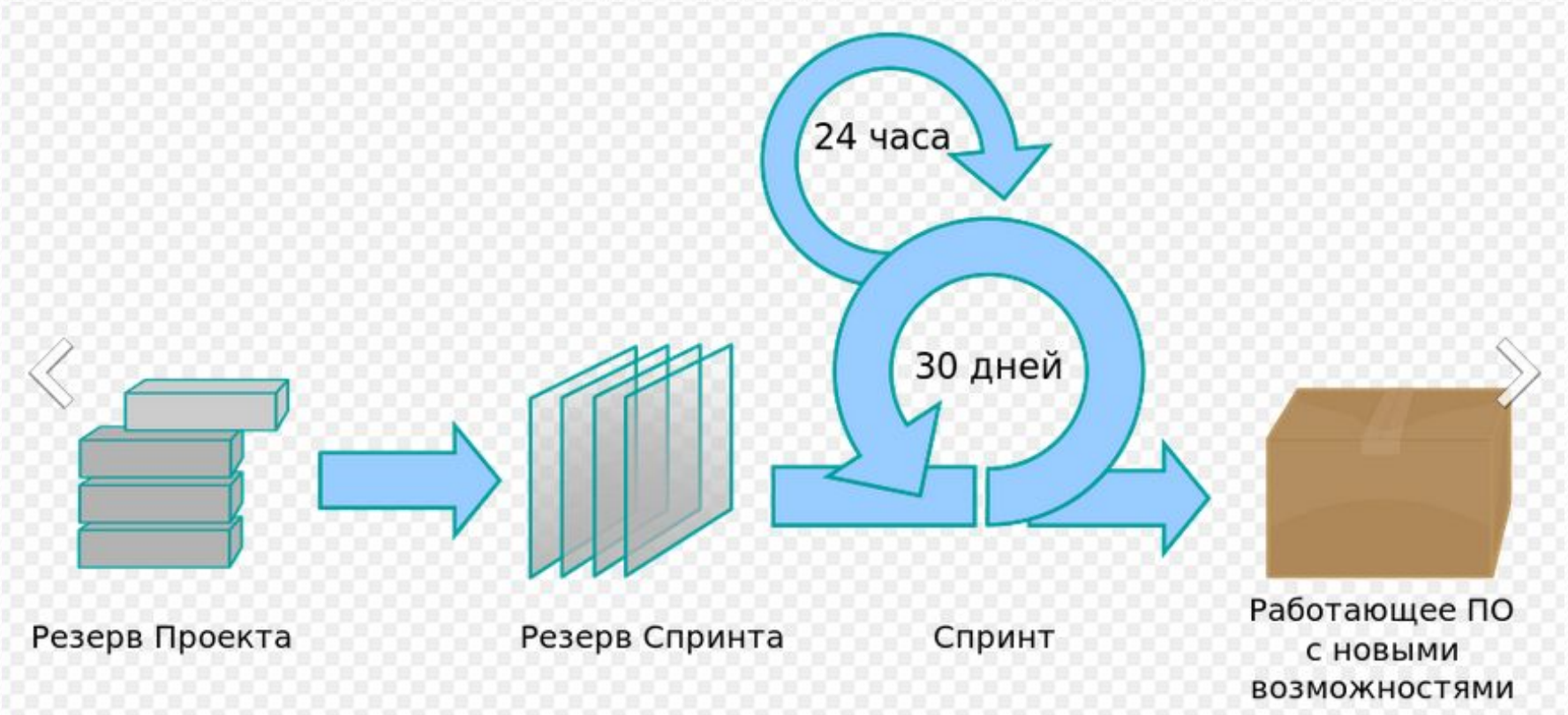
- Демонстрация

Происходит в конце спринта

SCRUM. Роли

- Основные (свиньи)
  - Владелец продукта (постановщик задачи)
  - Руководитель (Scrum Master)
  - Команда (!)
- Остальные (куры)
  - Пользователи
  - Клиенты
  - Эксперты-консультанты

Методология SCRUM



Методология SCRUM

Заказчик определяет и периодически меняет функциональные требования

Руководитель проекта расставляет приоритеты

Формируются небольшие группы (1-6, реже до 9) человек для реализации небольших частей проекта (спринт, отдельный набор работ)

Формируется sprint backlog проекта для каждой группы

Выполнение sprint происходит группой автономно. Руководитель не вправе влиять на sprint

Методология SCRUM

- Каждая группа ежедневно выполняет схватки (scrum) (10-30 мин) (разработчики демонстрируют динамику выполнения работ и проблемы). Каждый говорит:
  - Что сделано в предыдущий день?
  - Что будет сделано каждым в следующий день?
  - Что мешает работать или повышать производительность?
- Участвовать могут все, говорить только основные участники (Заказчик убеждается, что проект идет нормально, руководитель убеждается, что нет помех, мешающих решать проблемы)
- Задача руководителя группы – решать проблемы
- По окончании спринта – встреча с руководителями и заказчиками (демонстрация)

SCRUM. Выводы

Сейчас есть гибридные подходы (SCRUM и XP)

Может использоваться для относительно небольших проектов с меняющимися требованиями в основном для информационных систем и других объектно-ориентированных систем.

Внутри спринта – модель водопада.

3.3. Подходы исследовательского программирования

Исследовательское программирование имеет следующие особенности:

- разработчик ясно представляет направление поиска, но не знает заранее, как далеко он сможет продвинуться к цели;
- нет возможности предвидеть объем ресурсов для достижения того или иного результата;
- разработка не поддается детальному планированию, она ведется методом проб и ошибок;
- работа связана с конкретными исполнителями и отражает их личностные качества.

В основе исследовательского программирования в большей степени, чем в других подходах, лежит искусство.

3.3.1.Компьютерный дарвинизм

Название данного подхода было предложено Кеном Томпсоном (Ken Thompson). Подход основан на принципе восходящей разработки, когда система строится вокруг ключевых компонентов и программ, которые создаются на ранних стадиях проекта, а затем постоянно модифицируются. Все более крупные блоки собираются из ранее созданных мелких блоков.

Компьютерный дарвинизм представляет собой метод проб и ошибок, основанный на интенсивном тестировании. Причем на любом этапе система должна работать, даже если это минимальная версия того, к чему стремятся разработчики. Естественный отбор оставит только самое жизнеспособное.

Подход состоит из трех основных процессов:

- Макетирования (прототипирования).
- Тестирования.
- Отладки.

Одной из интересных особенностей подхода является максимально возможное распараллеливание процессов тестирования и отладки.

Рассмотренные технологические подходы

1. Подходы со слабой формализацией

2 Строгие (классические, жесткие, предсказуемые) подходы

- 2.1. Каскадные технологические подходы.
  - 2.1.1. Классический каскадный подход
  - 2.1.2. Каскадно-возвратный подход
  - 2.1.3. Каскадно-итерационный подход
  - 2.1.4. Каскадный подход с перекрывающимися процессами
  - 2.1.5. Каскадный подход с подпроцессами
  - 2.1.6. Спиральная модель
- 2.2. Каркасные подходы.
  - 2.2.1. Рациональный унифицированный процесс (RUP)
- 2.3. Генетические подходы.
  - 2.3.1. Синтезирующее программирование
  - 2.3.2. Сборочное (расширяемое) программирование
  - 2.3.3. Конкретизирующее программирование
- 2.4. Подходы на основе формальных преобразований.
  - 2.4.1. Технология стерильного цеха
  - 2.4.2. Формальные генетические подходы

3 Гибкие (адаптивные, легкие) подходы

- 3.1. Ранние технологические подходы быстрой разработки (RAD)
  - 3.1.1. Эволюционное прототипирование
  - 3.1.2.Итеративная разработка
  - 3.1.3. Постадийная разработка
- 3.2. Адаптивные подходы.
  - 3.2.1. Экстремальное программирование (XP)
  - 3.2.2. Адаптивная разработка
- 3.3. Подходы исследовательского программирования.
  - 3.3.1. Компьютерный дарвинизм

Технологические подходы к проектированию ПО. Итоги

	Класси-ческая	Прото-типиро-вание	Спи-ральная	Инкремент-ная	RAD	RUP	XP	Scrum
Стратегии (одно-кратная, эволюци-онная, инкремент-ная)	о	э	э	и	и	и+э	э	э
Вид методологии (прогнозирующий, адаптивный)	Пр	Пр	Пр	Пр	Пр	Пр	Ад	Ад
Размер команды	1…	<=10	1…	1…	1…	1…	<=10	<10
Продолжитель-ность проекта (вы-сокая, средняя, низкая)	Выс	Низк	Выс	Низк	Низк	Сред. Выс	Низк	Низк
Промежуточные версии	–	–	+/–	+	+/–	+	+	+
Направленность на ИС	–	–	–	–	+	+	+	+