*Слайд 1*

Лекция 2

ОСНОВНЫЕ ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

*Слайд 2*

**Учебные вопросы:**

**1. Подходы к обеспечению надежности программного обеспечения.**

2. Процессы проектирования программного обеспечения.

3. Концепции борьбы со сложностью программного обеспечения.

**4. Правильность проектирования.**

**5. Жизненный цикл ПО. Модели жизненного цикла ПО.**

*Слайд 3*

Разработка программы включает **задачи двух типов: проектирование и тестирование**.

*Термин* **проектирование**, *с точки зрения разработки программного обеспечения, в настоящее время употребляется в нескольких смыслах.*

*Во многих организациях смысл слова «проектирование» произвольно ограничивается начальным этапом работы над проектом, а для обозначения последующих этапов используются такие термины, как «реализация», «разработка», «программирование». К сожалению, нет общепринятого соглашения об употреблении этих слов, различные организации понимают под этими словами разные группы процессов, что приводит к путанице при попытке сравнить два проекта.*

В дальнейшем под понятием ***проектированием*** будем понимать определение внутренних свойств системы и детализацию ее внешних (видимых) свойств в соответствии с выданными заказчиком требованиями к программному обеспечению (исходными условиями задачи), которые, в свою очередь, подвергаются анализу.

*Это определение охватывает различные виды деятельности по созданию программного обеспечения, начиная с определения требований и целей и кончая написанием текста программы, но подразумевает, конечно, наличие различных стадий проектирования.*

# 1. Подходы к обеспечению надежности программного обеспечения

*Рассмотрим принципы обеспечения надежности ПО, общие для всех стадий проектирования.*

*Все принципы и методы обеспечения надежности в соответствии с их целью можно разбить на четыре группы:*

**1) предупреждение ошибок,**

**2) обнаружение ошибок,**

**3) исправление ошибок,**

**4) обеспечение устойчивости к ошибкам.**

К **первой группе** относятся принципы и методы, позволяющие минимизировать или вообще исключить ошибки. Методы **второй группы** сосредоточивают внимание на функциях самого программного обеспечения, помогающих выявлять ошибки. К **третьей группе** относятся функции программного обеспечения, предназначенные для исправления ошибок или их последствий. **Четвертая группа** – Устойчивость к ошибкам – это мера способности системы программного обеспечения продолжать функционирование при наличии ошибок.

*Слайд 4*

Рассмотрим каждую из этих групп более подробно.

1) Предупреждение ошибок

К этой группе относятся принципы и методы, цель которых — не допустить появления ошибок в готовой программе. *Большинство методов концентрируется на отдельных процессах перевода и направлено на предупреждение ошибок в этих процессах.*

Их можно разбить на **следующие категории**:

1. Методы, позволяющие справиться со сложностью, свести ее к минимуму, так как это — главная причина ошибок перевода.
2. Методы достижения большей точности при переводе.
3. Методы улучшения обмена информацией.
4. Методы немедленного обнаружения и устранения ошибок.

*Эти методы направлены на обнаружение ошибок на каждом шаге перевода, не откладывая до тестирования программы после ее написания.*

*Очевидно, что предупреждение ошибок — оптимальный путь к достижению надежности программного обеспечения. Лучший способ обеспечить надежность — прежде всего не допустить возникновения ошибок. Гарантировать отсутствие ошибок практически не возможно, поэтому другие три группы методов опираются на предположение, что ошибки все-таки будут.*

2) Обнаружение ошибок

*Если предполагать, что в программном обеспечении какие-то ошибки все же будут, то лучшая (после предупреждения ошибок) стратегия — включить средства обнаружения ошибок в само программное обеспечение.*

Большинство методов направлено по возможности на незамедлительное обнаружение сбоев. Немедленное обнаружение имеет **два преимущества:** можно минимизировать как влияние ошибки, так и последующие затруднения для человека, которому придется извлекать информацию об этой ошибке, находить ее место и исправлять.

*Слайд 5*

**3) Исправление ошибок**

Следующий шаг — методы исправления ошибок; после того как ошибка обнаружена, либо она сама, либо ее последствия должны быть исправлены программным обеспечением.

*Некоторые устройства способны обнаружить неисправные компоненты и перейти к использованию идентичных запасных. Аналогичные методы* ***неприменимы к программному обеспечению*** *вследствие глубоких внутренних различий между сбоями аппаратуры и ошибками в программах. Если некоторый программный модуль содержит ошибку, идентичные «запасные» модули также будут содержать ту же ошибку.*

*Другой подход* к исправлению ошибок связан с попытками восстановить разрушения, вызванные ошибками, *например искажения записей в базе данных или управляющих таблицах системы.* *Польза от методов борьбы с искажениями ограниченна, поскольку предполагается, что разработчик заранее предугадает несколько возможных типов искажений и предусмотрит программно реализуемые функции для их устранения.*

**4) Обеспечение устойчивости к ошибкам**

Методы этой группы ставят своей целью обеспечить функционирование программной системы при наличии в ней ошибок. Они разбиваются на три подгруппы:

- динамическая избыточность,

- методы отступления,

- методы изоляции ошибок.

Истоки концепции динамической избыточности лежат в проектировании аппаратного обеспечения.

**Один из подходов к динамической избыточности - метод голосования.** *Данные обрабатываются независимо несколькими идентичными устройствами, и результаты сравниваются. Если большинство устройств выработало одинаковый результат, этот результат и считается правильным. В случае с программным обеспечением копия одного модуля всегда дает один и тот же результат, поэтому необходимо использовать неидентичные копии модуля (реализуют различные алгоритмы, созданы разными разработчиками, имеющие разные версии).*

***Дополнительное ПО повышает уровень сложности систем, что противоречивой идее предупреждения ошибок – минимизации сложности.***

**Второй подход к динамической избыточности** — выполнять эти запасные копии только тогда, когда результаты, полученные с помощью основной копии, признаны неправильными. *Если это происходит, система автоматически вызывает запасную копию. Если и ее результаты неправильны, вызывается другая запасная копия и т. д.*

*Этот подход страдает большинством перечисленных ранее недостатков. Кроме того, вполне вероятно, что если ресурсы при работе над проектом фиксированы, то при реализации «запасных» версий проектированию и тестированию будет уделено меньше внимания, чем если бы реализовывалась лишь одна копия и динамическая избыточность не использовалась.*

**Вторая подгруппа методов обеспечения устойчивости к ошибкам называется методами отступления или сокращенного обслуживания.** Эти методы приемлемы обычно лишь тогда, когда для системы программного обеспечения существенно важно благоприятно закончить работу.

*Например, если ошибка оказывается в системе, управляющей технологическими процессами, и в результате эта система выходит из строя, то может быть загружен и выполнен особый фрагмент программы, призванный подстраховать систему и обеспечить безаварийное завершение всех управляемых системой процессов. Аналогичные средства часто необходимы в операционных системах. Если операционная система обнаруживает, что она вот-вот выйдет из строя, она может загрузить аварийный фрагмент, ответственный за оповещение пользователей у терминалов о предстоящем сбое и за сохранение всех критических для системы данных.*

**Последняя подгруппа — методы изоляции ошибок.**

Основная их идея — не дать последствиям ошибки выйти за пределы как можно меньшей части системы программного обеспечения, так чтобы если ошибка возникнет, то не вся система оказалась неработоспособной; отключаются лишь отдельные функции в системе либо некоторые ее пользователи. *Например, во многих операционных системах изолируются ошибки отдельных пользователей, так что сбой влияет лишь на некоторое подмножество пользователей, а система в целом продолжает функционировать.* В телефонных системах для восстановления после ошибки, чтобы не рисковать выходом из строя всей системы, просто разрывают телефонную связь.

*Другие методы* *изоляции ошибок связаны с защитой каждой из программ в системе от ошибок других программ. Например, ошибка в прикладной программе, выполняемой под управлением операционной системы, должна оказывать влияние только на эту программу. Она не должна сказываться на операционной системе или других программах, функционирующих в этой системе.*

**Из этих трех подгрупп методов обеспечения устойчивости к ошибкам только третья, *изоляция ошибок*, применима для большинства систем программного обеспечения.**

*Важное обстоятельство, касающееся всех рассмотренных четырех подходов, состоит в том, что* ***обнаружение, исправление ошибок и устойчивость к ошибкам*** *в некотором отношении противоположны методам предупреждения ошибок.*

*В частности, обнаружение, исправление и устойчивость требуют дополнительных функций от самого программного обеспечения. Тем самым не* ***только увеличивается сложность*** *готовой системы,* ***но и появляется возможность внести новые ошибки при реализации этих функций****. Как правило,* **все** *рассматриваемые* ***методы предупреждения*** *и* ***многие методы обнаружения ошибок*** *применимы к любому программному проекту.* ***Методы исправления ошибок и обеспечения устойчивости применяются не очень широко.*** *Это зависит от области применения приложения. Если рассматривается, скажем, система реального времени, то ясно, что она должна сохранить работоспособность и при наличии ошибок, а тогда могут оказаться желательными и методы исправления и обеспечения устойчивости. К системам такого типа относятся телефонные системы, системы управления технологическими процессами, аэрокосмические и авиационные диспетчерские системы и операционные системы широкого назначения.*

*Слайд 6*

# 2. Процессы проектирования программного обеспечения

*Проектирование любого программного продукта включает несколько различных процессов.*

*При хорошей организации работы над проектом эти процессы явно выражены, так что могут быть установлены контрольные сроки, выбрана методология и по завершении каждого процесса можно проверить качество полученных результатов.*

*Напротив, при плохом руководстве некоторые из этих процессов или все они не выделяются явно: каждый процесс по-прежнему присутствует в каком-то виде, но некоторые из них проходят неявно, вследствие чего контрольные сроки, методология и оценки никогда не устанавливаются.*

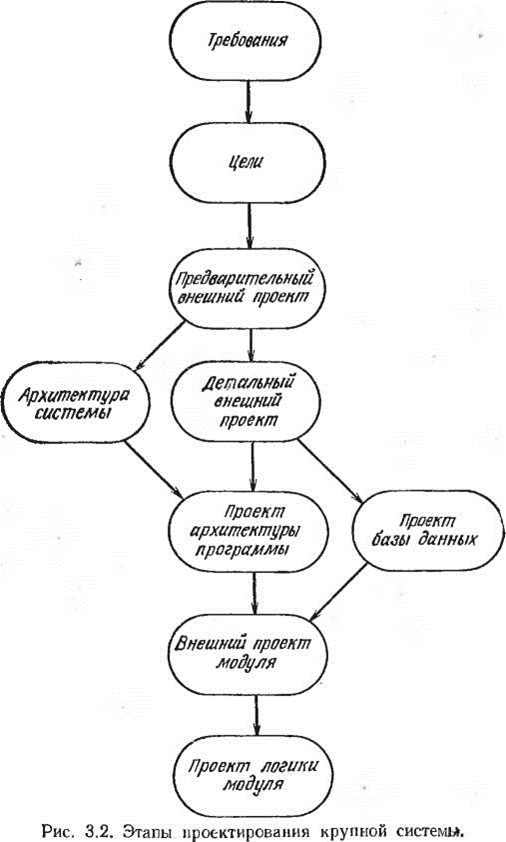


Рис. 1 Модель процессов проектирования

На рис. 1 представлена модель процессов проектирования типичной крупной программной системы. *Отметим, что модель не зависит от методологии. Все указанные в ней действия должны выполняться в той или иной форме во всякой разработке, независимо от того, какой язык программирования был принят, писал ли пользователь исходные требования, использовалось ли «структурное программирование» и т. д.*

**1) На первом шаге** составляется перечень требований, т. е. четкое определение того, что пользователь ожидает от готового продукта.

**2) Следующий шаг** касается **постановки целей** — задач, которые ставятся перед окончательным результатом и самим проектом.

**3) Затем выполняется** внешний проект высокого уровня. На этом шаге определяется взаимодействие с пользователем, но не рассматриваются многие его детали, такие, как форматы ввода-вывода.

*Исходный внешний проект приводит к двум параллельным процессам.*

А) В процессе *детального внешнего проектирования* завершается определение взаимодействия с пользователем, описываются его мельчайшие подробности.

Б) В процессе разработки **архитектуры системы** выполняется разложение ее на множество программ, подсистем или компонент и определяются сопряжения между ними.

**4) Эти два шага** ведут к процессу **проектирования архитектуры (структуры) программы**, в котором проектируются модули, их сопряжения и взаимосвязи для каждой программы, компоненты или подсистемы.

**5) Следующий процесс — внешнее проектирование модуля** — это точное определение всех сопряжений модуля.

**6) Последний шаг — проектирование логики модуля** — состоит в разработке внутренней логики каждого модуля системы, он включает также выражение этой логики текстом конкретной программы.

**7)** Последний изображенный на рис. 1 процесс — **проектирование базы данных.** Это процесс определения всех внешних для программной системы структур данных, например записей в файле или в базе данных.

*На рис. 1 представлены основные зависимости между процессами проектирования, показано, какие процессы проектирования должны быть полностью закончены, прежде чем может быть завершен другой процесс.* *Например, необходимо закончить внешнее проектирование всех модулей системы до завершения проектирования логики модулей.* *Однако проектирование логики отдельных модулей, конечно, можно завершить и до окончания внешнего проектирования всех модулей.*

*Необходимо отметить, что не* должно *быть никакого перекрытия между несмежными процессами.* *Например, процесс проектирования логики модулей не следует начинать до окончания проектирования структуры программы.* *И второй пример: не следует приступать к разработке архитектуры системы до того, как полностью будут определены ее цели.*

*Следует также помнить, что в* ***работе над любым проектом последовательность процессов проектирования отнюдь не так проста****. Есть существенная обратная связь между процессами. Например, во время одного из шагов внешнего проектирования могут быть обнаружены погрешности в формулировке целей, тогда нужно немедленно вернуться и исправить их. При проектировании структуры программы можно внезапно обнаружить, что указанная во внешних спецификациях функция неосуществима или обойдется слишком дорого, тогда может понадобиться принять компромиссное решение и изменить внешние спецификации.*

*Помимо этого модель, иллюстрированная рис. 1, с некоторыми изменениями, может служить моделью и для малых проектов.* *Когда проектируется единственная программа, то часто отсутствуют процессы проектирования архитектуры системы и проектирования базы данных; процессы исходного и детального внешнего проектирования также зачастую сливаются воедино.*

*Слайд 7*

# 3. Концепции борьбы со сложностью программного обеспечения

*Сложность — это основная причина ошибок перевода и, следовательно, одна из главных причин ненадежности программного обеспечения.*

*Сложность почти не поддается ни точному определению, ни измерению.*

**Мерой сложности** объекта является количество интеллектуальных усилий, необходимых для понимания этого объекта.

В борьбе со сложностью программного обеспечения можно привлечь три концепции.

**Первая концепция — независимость.**

В соответствии с этой концепцией для минимизации сложности необходимо максимально усилить независимость компонент системы. *По существу, это означает такое разбиение системы, чтобы высокочастотная динамика ее была заключена в единых компонентах, а межкомпонентные взаимодействия представляли лишь низкочастотную динамику системы.*

**Вторая концепция — иерархическая структура.**

Иерархия позволяет разделить систему по уровням понимания. Каждый уровень представляет собой совокупность структурных отношений между элементами нижних уровней. Концепция уровня позволяет понять систему, скрывая несущественные уровни детализации.

**Иерархия позволяет проектировать, описывать и понимать сложные системы.**

**Третья концепция – проявление связей всюду, где они возникают.** Основная проблема многих больших программных систем — огромное количество независимых побочных эффектов, создаваемых компонентами системы. *Из-за этих побочных эффектов систему невозможно понять, невозможно в ней разобраться, и соответственно, такую систему было очень трудно спроектировать хотя бы с минимальной гарантией надежности.*

*Слайд 8*

# 4. Правильность проектирования

*Явно выделенным этапом всякого процесса проектирования должна быть проверка правильности результатов, т. е. попытка найти ошибки перевода, возникшие в этом процессе. Основания для этого иллюстрирует рис. 2.*

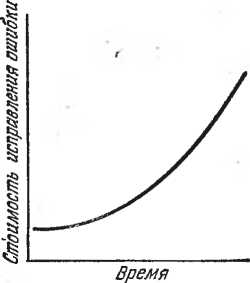
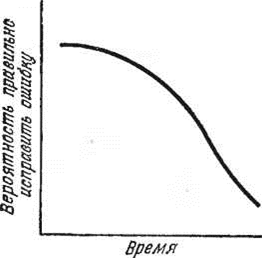
 

Рис. 2. Основания для раннего обнаружения ошибок проектирования.

*Дело не только в том, что стоимость исправления ошибки тем ниже, чем раньше она будет обнаружена, но и в том, что вероятность правильно исправить ошибку на ранней стадии работы над проектом значительно выше, чем в случае, если ошибка обнаружена на более поздних этапах, например при тестировании системы.*

Хотя проверка правильности каждого отдельного этапа проектирования проводится по разным методикам, можно сформулировать общую систему правил проверки в виде правила **«n плюс-минус один»**.

Вопросом первостепенной важности при проверке правильности проекта является привлечение компетентных специалистов.

Компетентные специалисты – это те, в чьих интересах обнаружить все ошибки. *Пусть, например, закончен n-й этап проектирования архитектуры системы. Проектировщики предыдущего этапа* ***n–1*** *– это авторы исходных внешних спецификаций, а проектировщики этапа* ***n+1*** *– это разработчики структуры программы. Именно им и следует доверить тестирование архитектуры системы.*

Правило ***«n плюс-минус один»*** состоит в следующем: проверка правильности этапа проекта должна осуществляться проектировщиками этапов ***n+1*** и ***n–1***.

Следует отметить, что в случае необходимости должны быть установлены формальные процедуры, обеспечивающие ее применение.

**Цель** должна состоять в том, чтобы обнаружить любой мыслимый дефект, любую слабость в проекте системы, а не в том, чтобы просмотреть проект и показать, что он правилен. *Заинтересованность проектировщика фазы* п*—1 (автора внешних спецификаций) в нахождении ошибок заключается в нахождении ошибок системных архитекторов, внесших ошибки в спецификации.*

*Разработчик фазы n+1 в такой же мере заинтересован в обнаружении ошибок, хотя и по другой причине. Его заинтересованность в обнаружении ошибок заключается в нахождении дефектов, внесенных на предыдущем этапе, т.к. в последующем при реализации этапа n+1 именно эти ошибки могут иметь первостепенное значение.*

*Слайд 9*

*При наличии ошибки появляется необходимость во внесении изменений.* Изменение в проекте – объективный фактор разработки программного обеспечения. Опытный разработчик программного обеспечения помнит об этом; он заранее запланировал изменения, организовал свой проект так, что изменение не становится травмирующим происшествием, он знает, как внести изменения таким образом, чтобы не понизить надежность работы системы.

**Первое правило работы с изменениями** – во время всего цикла работы над проектом поддерживать документацию на уровне последних решений.

**Второе правило** – в какой-то определенный момент времени «заморозить», приостановить результаты каждого процесса проектирования. Приостановка процесса проектирования не означает, что больше нельзя вносить изменения; предполагается лишь, что с этого момента изменения должны проходить формальную процедуру утверждения.

**Третье правило работы с изменениями** – проверять правильность всякого изменения в такой же степени, в какой проверялось исходное решение.

**Четвертое правило** – убедиться, что все нужные изменения сделаны на всех уровнях. Программист, проектирующий внутреннюю логику модуля, может сделать изменения, но не заметить при этом, что они влекут изменения внешних характеристик. Программное обеспечение и спецификации теперь рассогласованы, а это означает ошибку в программном обеспечении.

*Слайд 10*

**5.** **Жизненный цикл ПО. Модели жизненного цикла ПО.**

Жизненный цикл ПО

*Одним из базовых понятий методологии проектирования ПС является понятие жизненного цикла программного обеспечения (ЖЦ ПО).* **ЖЦ ПО** - **это непрерывный процесс**, **который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации**.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207:1995 «Information Technology - Software Life Cycle Processes (ISO - International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО. В данном стандарте **ПО определяется как набор компьютерных программ**, **процедур и**, **возможно**, **связанных с ними документации и данных**. В России имеется система стандартов, регламентирующих создание ПО - ГОСТ ЕСПД.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

* **Пять основных процессов ЖЦ ПО** (приобретение, поставка, разработка, эксплуатация, сопровождение);
* **Восемь вспомогательных процессов**, обеспечивающих выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, решение проблем);

- **Четыре организационных процесса** (управление проектами, создание инфраструктуры проекта, усовершенствование, обучение).

*Слайд 11*

Основные процессы ЖЦ ПО

**Процесс приобретения** состоит из действий и задач заказчика, приобретающего ПО.

**Процесс поставки** охватывает действия и задачи, выполняемые поставщиком, который снабжает заказчика программным продуктом или услугой.

**Процесс разработки** включает в себя все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала и т.д.

*Разработка ПО включает в себя:*

* *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.*

**Процесс эксплуатации** включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т. д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

**Процесс сопровождения** предусматривает действия и задачи, выполняемы сопровождающей организацией. *Под сопровождением понимается внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.*

*Слайд 12*

Вспомогательные процессы ЖЦ ПО

**Процесс документирования** предусматривает формализованное описание информации, созданной в течение ЖЦ ПО.

**Процесс управления конфигурацией** позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. *Общие принципы и рекомендации конфигурационного учета, планирования и управления конфигурациями ПО отражены в проекте стандарта ISO 12207-2.*

**Процесс обеспечения качества** обеспечивает соответствующие гарантии того, что ПО и процессы его ЖЦ соответствуют требованиям и утвержденным планам.

**Процесс верификации** состоит в определении того, что программные продукты, являющиеся результатами некоторого действия, полностью удовлетворяют требованиям или условиям, обусловленным предшествующими действиями. *Может выполняться как разрабатывающей, так и независимой организацией.*

**Процесс аттестации** предусматривает определение полноты соответствия заданных требований и созданной системы или программного продукта их конкретному функциональному назначению. *Может выполняться как разрабатывающей, так и независимой организацией.*

**Процесс совместной оценки** предназначен для оценки состояния работ по проекту и ПО, создаваемого при выполнении работ. *Он сосредоточен в основном на контроле планирования и управления ресурсами, персоналом, аппаратурой и инструментальными средствами проекта. Может выполняться двумя сторонами, участвующими в договоре, при этом одна из сторон проверяет другую.*

**Процесс аудита** представляет собой определение соответствия требованиям, планам и условиям договора. *Аудит - это ревизия, проводимая компетентным органом в целях обеспечения независимой оценки степени соответствия ПО или процессов установленным требованиям.*

**Процесс разрешения проблем** предусматривает анализ и решение проблем независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения и других процессов.

*Слайд 13*

Организационные процессы ЖЦ ПО

**Процесс управления** состоит из действий и задач, которые могут выполняться любой стороной, управляющей своими процессами. Включает следующие действия: инициирование управления, планирование, выполнение и контроль, проверку и оценку, завершение.

**Процесс создания инфраструктуры** охватывает выбор и поддержку технологии, стандартов и инструментальных средств, выбор и установку аппаратных и программных средств, используемых для разработки, эксплуатации и сопровождения ПО.

**Процесс усовершенствования** предусматривает оценку, измерение, контроль и усовершенствование процессов ЖЦ ПО. *Направлено на повышение производительности труда всех участвующих в нем специалистов за счет совершенствования технологии, методов управления, выбора инструментальных средств и обучения персонала.*

**Процесс обучения** охватывает первоначальное обучение и последующее постоянное повышение квалификации персонала.

*Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результатами анализа, в частности, являются функциональные модели, информационные модели и соответствующие им диаграммы.* ЖЦ ПО носит итерационный характер: результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах.

*Слайд 14*

Модели жизненного цикла ПО

*Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО (под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ПС и специфики условий, в которых последняя создается и функционирует). Регламенты стандарта являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.*

К настоящему времени известны следующие основные модели ЖЦ:

1) **каскадная модель** (классический жизненный цикл);

2) **итерационная модель** ЖЦ ПС;

3) **инкрементная модель;**

4) **спиральная модель**;

5) **рациональный унифицированный процесс;**

6) **MSF (Microsoft Solution Framework).**

*Слайд 15*

**Каскадная модель**   
(классический жизненный цикл)

*В изначально существовавших однородных ПС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ.* Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 3). *Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.*

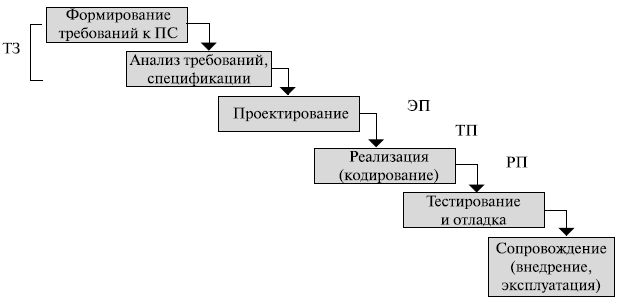


Рис. 3. Каскадная схема разработки ПО

**Достоинства** применения каскадного подхода заключаются в следующем:

**1)** *на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;*

**2)** *выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.*

Недостатки каскадной модели:

**1)** *выявление и устранение ошибок производится только на стадии тестирования, которое может существенно растянуться;*

**2)** *реальные проекты часто требуют отклонения от стандартной последовательности шагов;*

**3)** *цикл основан на точной формулировке исходных требований к ПС, реально в начале проекта требования заказчика определены лишь частично;*

**4)** *результаты работ доступны заказчику только по завершении проекта.*

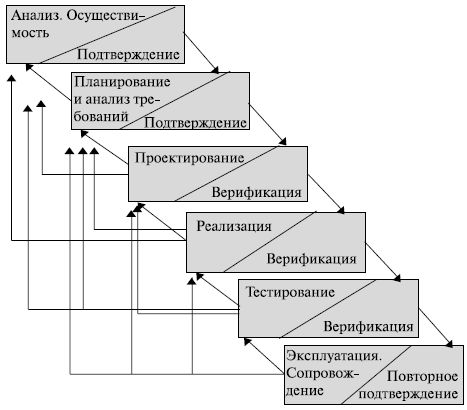
*Каскадный подход хорошо зарекомендовал себя при построении ПС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи.*

*Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ПС "заморожены" в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО, пользователи получают систему, не удовлетворяющую их потребностям. Модели (как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением.*

*Слайд 16*

**Итерационная модель ЖЦ ПС**

*С ростом коммерческих проектов выяснилось, что не всегда удается детально проработать проект будущей системы, поскольку многие аспекты ее функционирования в динамических сферах деятельности (бизнес) меняются, пока система создается.* ***Потребовалось изменить процесс разработки так, чтобы гарантировать внесение необходимых исправлений после завершения какого-либо этапа разработки.*** *Так появилась итерационная модель ЖЦ ПС, называемая* ***моделью с промежуточным контролем или моделью с циклическим повторением фаз****.*



*Рис. 4. Итерационная модель жизненного цикла программного обеспечения*

В итерационной модели (см. рис.4) недостатки проектирования и программирования могут быть устранены позже путем частичного возврата на предыдущую стадию. *Чем ниже уровень обнаружения ошибки, тем дороже ее исправление. Если стоимость усилий, необходимых для обнаружения и устранения ошибок на стадии написания кода, принять за единицу, то стоимость выявления и устранения ошибки на стадии выработки требований будет в 5-10 раз меньше, а стоимость выявления и устранения ошибки на стадии сопровождения – в 20 раз больше (рис.5).*



*Рис. 5. Зависимость стоимости и устранения ошибки от стадии разработки ПО*

*Слайд 17*

**Достоинства:**

**1)** затраты, которые получаются в связи с изменением требований пользователей, уменьшаются, повторный анализ и совокупность документации значительно сокращаются по сравнению с каскадной моделью;

**2)** 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.

**3)** 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:**

**1)** менеджеры должны постоянно измерять прогресс процесса. в случае быстрой разработки не стоит создавать документы для каждого минимального изменения версии;

**2)** структура системы имеет тенденцию к ухудшению при добавлении новых компонентов — постоянные изменения нарушают структуру системы. *Чтобы избежать этого требуется дополнительное время и деньги на рефакторинг. Плохая структура делает программное обеспечение сложным и дорогостоящим для последующих изменений. А прерванный Жизненный цикл ПО приводит еще к большим потерям.*

*Схема не позволяет оперативно учитывать возникающие изменения и уточнения требований к ПО. Согласование результатов разработки с пользователями производится только в точках, планируемых после завершения каждого этапа работ, а общие требования к ПО зафиксированы в виде технического задания на всё время её создания. Таким образом, пользователи зачастую получаю ПП, не удовлетворяющий их реальным потребностям.*

*Слайд 18*

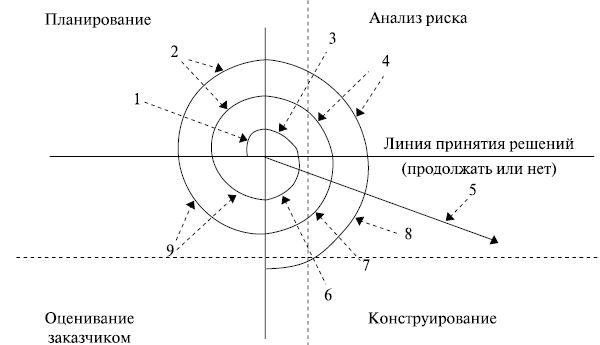
**Спиральная модель**

*Для преодоления перечисленных проблем была Барри Боэмом в 1986 году предложена спиральная модель ЖЦ (рис. 6), делающая упор на начальные этапы ЖЦ:* ***анализ и проектирование,*** где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов.

*Спиральная модель – классический пример применения эволюционной стратегии конструирования. Модель (автор Б. Боэм, 1988) базируется на лучших свойствах классического жизненного цикла и макетирования, к которым добавляется новый элемент –* ***анализ риска, отсутствующий в этих парадигмах****.*

**Жизненный цикл** — на каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка.

Модель определяет четыре действия, представляемые четырьмя квадрантами спирали (рис.6).



**Рис. 6.** Спиральная модель жизненного цикла программного обеспечения

1. Планирование – определение целей, вариантов и ограничений.
2. Анализ риска – анализ вариантов и распознавание/выбор риска.
3. Конструирование – разработка продукта следующего уровня.
4. Оценивание – оценка заказчиком текущих результатов конструирования.

**Под рисками (по приоритетам)** **в спиральной модели понимается:**

1. Дефицит специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация несоответствующей функциональности.
4. Разработка неправильного пользовательского интерфейса.
5. «Золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей.
6. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. Разрыв между квалификацией специалистов и требованиями проекта

*Интегрирующий аспект* *спиральной модели очевиден при учете радиального измерения спирали.* С каждой итерацией по спирали строятся все более полные версии ПС. В первом витке спирали определяются начальные цели, варианты и ограничения, распознается и анализируется риск. *Если анализ риска показывает неопределенность требований, на помощь разработчику и заказчику приходит* ***макетирование****.*

*Для дальнейшего определения проблемных и уточненных требований может быть использовано моделирование. Заказчик оценивает инженерную (конструкторскую) работу и вносит предложения по модификации* ***(квадрант оценки заказчиком).*** *Следующая фаза планирования и анализа риска базируется на предложениях заказчика. В каждом цикле по спирали результаты анализа риска формируются в виде "продолжать, не продолжать". Если риск слишком велик, проект может быть остановлен.*

*Слайд 19*

*В большинстве случаев движение по спирали продолжается, с каждым шагом продвигая разработчиков к более общей модели системы. В каждом цикле по спирали требуется конструирование (нижний правый квадрант), которое может быть реализовано классическим жизненным циклом или макетированием.* Заметим, что количество действий по разработке (происходящих в правом нижнем квадранте) возрастает по мере продвижения от центра спирали.

Эти действия пронумерованы на рис.6 и имеют следующее содержание:

1. – начальный сбор требований и планирование проекта;
2. – та же работа, но на основе рекомендаций заказчика;
3. – анализ риска на основе начальных требований;
4. – анализ риска на основе реакции заказчика;
5. – переход к комплексной системе;
6. – начальный макет системы;
7. – следующий уровень макета;
8. – сконструированная система;
9. – оценивание заказчиком.

*Слайд 20*

**Достоинства** **спиральной модели:**

1. позволяет быстрее показать пользователям системы работоспособный продукт, *тем самым, активизируя процесс уточнения и дополнения требований;*
2. допускает изменение требований при разработке программного обеспечения, *что характерно для большинства разработок, в том числе и типовых;*
3. 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;*
4. позволяет получить более надежную и устойчивую систему. *По мере развития программного обеспечения ошибки и слабые места обнаруживаются и исправляются на каждой итерации;*
5. эта модель разрешает пользователям активно принимать участие при планировании, анализе рисков, разработке, а также при выполнении оценочных действий;
6. уменьшаются риски заказчика. *Заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта;*
7. обратная связь по направлению от пользователей к разработчикам выполняется с высокой частотой и на ранних этапах модели, *что обеспечивает создание нужного продукта высокого качества.*

**Недостатки** **спиральной 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 стадии.

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.*

*Слайд 21*

**Область применения спиральной модели**

1) при разработке проектов, использующих новые технологии;

2) при разработке новой серии продуктов или систем;

3) при разработке проектов с ожидаемыми существенными изменениями или дополнениями требований;

4) для выполнения долгосрочных проектов;

5) при разработке проектов, требующих демонстрации качества и версий системы или продукта через короткий период времени;

6) при разработке проектов, для которых необходим подсчет затрат, связанных с оценкой и разрешением рисков.

*Слайд 22*

# Литература

1. Казарин О. В. Надежность и безопасность программного обеспечения: учебное пособие для бакалавриата и магистратуры /О. В. Казарин, И. Б. Шубинский. — М.: Издательство Юрайт, 2018. — 342 с.
2. Орлов С.А. Технологии разработки программного обеспечения : учеб. для вузов. - 3-е изд СПб.: Питер, 2004
3. Аллен Э. Типичные ошибки проектирования : [Пер. с англ.] СПб.: Питер, 2003
4. Фатрелл Р.Т. Управление программными проектами: достижение оптимального качества при минимуме затрат : [Пер. с англ.] М.: Вильямс, 2003
5. Липаев В.В. Надежность программных средств./ В.В. Липаев — М.: СИНТЕГ,1998
6. Майерс Г. Надежность программного обеспечения: Пер. с англ/ Г. Майерс.. - М.: Мир, 1980.

*Слайд 23*

# Контрольные вопросы

1. Какие известны принципы и методы обеспечения надежности?
2. Какие категории включает в себя принцип «Предупреждение ошибок»?
3. Какие методы включает в себя принцип «Обеспечение устойчивости к ошибкам»?
4. Какие концепции используются для борьбы со сложностью программного обеспечения?
5. С какой целью применяется правило «n плюс-минус один»?
6. Дайте определение понятию «жизненный цикл ПО».
7. Сколько групп процессов предусматривает стандарт ISO/IEC 12207, назовите их.
8. Какие модели ЖЦ ПО известны?
9. Дайте характеристику каскадной модели ЖЦ ПО, назовите её достоинства и недостатки.
10. Дайте характеристику спиральной модели ЖЦ ПО, назовите её достоинства и недостатки.