**Методология разработки критического ПО**

**Введение**

Тема критического программного обеспечения (ПО) является важной и актуальной по следующим причинам:

* *повсеместность программного обеспечения*– ПО все большего объема устанавливается в бытовых устройствах, а сбои напрямую затрагивают жизнь обычных людей;
* *критичность программного обеспечения*– по мере того, как программное обеспечение все глубже проникает в структуру общества, сбои все чаще становятся критичными, даже если само ПО не является сложным;
* участие людей – с внедрением программного обеспечения в управляющие системы, в работе которых участвуют люди, все более важной становится проблема взаимодействия человека с программами;
* *запутанность*– зависимости между различными компонентами программного обеспечения усложняются;
* *возрастающая скорость эволюции*– скорость эволюции ПО увеличивается по мере роста требований со стороны пользователей (рынок программного обеспечения безжалостен к поставщикам, не способным дешево и быстро изменять свое программное обеспечение).

Для разработки такого вида ПО требуется хорошо спланированный процесс управления проектом со всеми наборами совершенствующихся методов, методологий и инструментальных средств.

Одним из ключевых понятий управления проектами, в том числе в приложении к индустрии программного обеспечения, является жизненный цикл (ЖЦ) проекта (Project Life Cycle Management - PLCM).

Одним из этапов ЖЦ является процесс разработки, который определяет работы и задачи разработчика, включающий в себя процессы анализа требований, проектирования архитектуры с использованием разных принципов (объектного, компонентного и др.) и техник: абстракции, декомпозиции, инкапсуляции и т.д., испытаний ПО.

В данной работе будут рассмотрены следующие важные методы и методологии в процессе разработки критического ПО.

Раздел 1 посвящен процессам модели процессов ЖЦ ПО (стандарт ISO/IEC/IEEE 12207:2008 [1]), имеющими отношение к разработке ПО. В Разделе 2 рассмотрим методы декомпозиции ПО как объекта разработки при реализации принципа «разделяй и властвуй». В Разделе 3 предоставлены ключевые понятия в архитектуре ПО такие, как описание архитектуры, критерии «сцепления-связанности». Раздел 4 посвящен тестированию и квалификационным испытаниям ПО, обосновывает их значимость в разработке критического ПО.

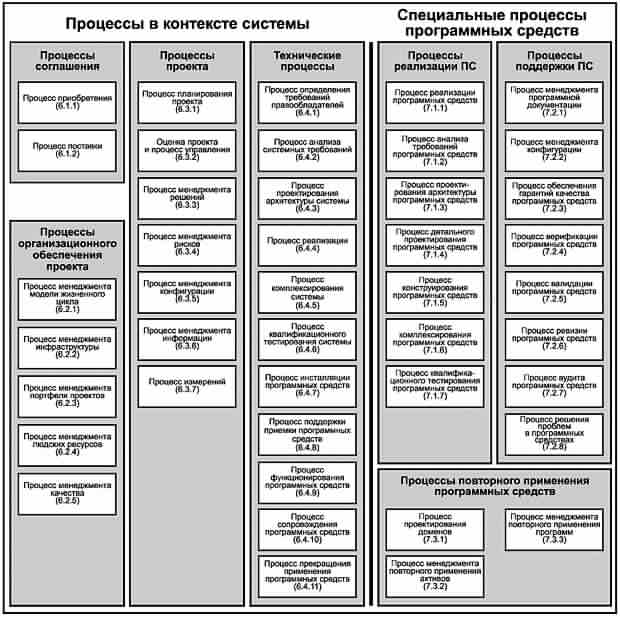
**Раздел 1**

**1.1 Разработка и ЖЦ ПО**

В ходе разработки программного обеспечения используются следующие процессы стандарта ISO/IEC/IEEE 12207:2008 [1]:

* Процесс определения требований правообладателей;
* Процесс анализа системных требований;
* Процесс проектирования архитектуры системы;
* Процесс квалификационного тестирования системы;
* Процесс реализации ПО;
* Процесс анализа требований к программным средствам;
* Процесс проектирования архитектуры программных средств;
* Процесс детального проектирования программных средств;
* Процесс конструирования программных средств;
* Процесс квалификационного тестирования программных средств.

Каждый из этих процессов относится к определенной группе процессов, представленных на рисунке ниже.



*Рисунок 1 – Группы процессов жизненного цикла [1]*

На этапе разработки программных средств нас интересуют группы технических процессов и процессов реализации ПО.

Остановимся детально на каждом из перечисленных выше процессов.

**1.2 Описание процессов этапа разработки ПО**

1. Процесс определения требований правообладателей

Цель процесса определения требований правообладателей состоит в выявлении требований к системе, выполнение которых может обеспечивать предоставление услуг, необходимых пользователям и другим правообладателям в заданной среде применения.

Этот процесс позволяет определять правообладателей или классы правообладателей, которые связаны с системой на протяжении всего ее жизненного цикла, а также их потребности и пожелания. В рамках процесса они анализируются и преобразуются в общую совокупность требований правообладателей, которые описывают желаемое поведение системы в процессе взаимодействия со средой применения. Она служит в качестве ссылки, по отношению к которой каждая предоставляемая услуга подвергается валидации для подтверждения того, что система полностью удовлетворяет заявленным требованиям.

В результате успешного осуществления процесса определения требований правообладателей:

a) задаются требуемые характеристики и условия использования услуг;

b) определяются ограничения для системных решений;

c) достигается возможность прослеживания от требований правообладателей к правообладателям и их потребностям;

d) описывается основа для определения системных требований;

e) определяется основа для валидации соответствия услуг;

f) формируется основа для ведения переговоров и заключения соглашений о поставке услуги или продукции.

**Виды деятельности и задачи**

При реализации проекта должны осуществляться следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса определения требований правообладателя.

Идентификация правообладателей

Данный вид деятельности состоит из решения следующей задачи:

* При реализации проекта необходимо идентифицировать отдельных правообладателей или классы правообладателей, имеющих законный интерес к системе в течение ее жизненного цикла. К ним относятся, по крайней мере: пользователи, операторы, организации поддержки, разработчики, производители, обучающие организации, организации технического обслуживания и ремонта, организации, распоряжающиеся ресурсами, приобретающие стороны и организации поставщика, стороны, ответственные за интерфейс с внешними объектами, регулирующие органы и представители общественности. В случае, если непосредственная коммуникация неосуществима (например, для потребительских товаров и услуг), выбираются представители или доверенные лица правообладателей.

Идентификация требований

Данный вид деятельности состоит из решения следующих задач:

* Должны быть выявлены требования правообладателей проекта. Требования правообладателей могут выражаться в форме потребностей, пожеланий, требований, ожиданий и воспринятых ограничений отдельных правообладателей, которые, в свою очередь, выражаются в терминах модели (текстовой или формализованной), ориентированной на цели и поведение системы и описывающей ее в контексте среды и условий функционирования. Для осуществления этих действий может быть полезной модель качества продукции и требований к качеству. В требованиях правообладателей должны учитываться нужды, потребности общества и ограничения, налагаемые приобретающей организацией, а также возможностями и способностями пользователей и оперативного персонала. Рекомендуется ссылаться на источники, например, на ходатайства или соглашения, их законность и обоснования, а также на допущения правообладателей и значение, которое правообладатели придают выполнению своих требований. Для потребностей ключевых правообладателей необходимо устанавливать показатели результативности, определенные таким образом, чтобы эксплуатационные характеристики могли быть измерены и оценены. Если значительные риски являются вероятным результатом возникающих вопросов (т.е. потребностей, пожеланий, ограничений, пределов, обеспокоенности, препятствий, факторов или соображений), имеющих отношения к людям (пользователям и другим правообладателям) и их вовлечению или взаимодействию с системой на любом отрезке времени в процессе жизненного цикла этой системы.
* В проекте необходимо определять ограничения системных решений, которые являются неизбежным следствием существующих соглашений, управленческих и технических решений.
* В проекте необходимо определять представительную совокупность последовательности видов деятельности для идентификации всех требуемых услуг, соответствующих ожидаемым рабочим сценариям и сценариям поддержки в заданных условиях применения.
* При реализации проекта необходимо определять взаимодействие между пользователями и системой, принимая во внимание способности человека и ограниченность его навыков.
* В проекте необходимо установить требования к здоровью, безопасности, защищенности, окружающим условиям и другие требования и функции правообладателей, имеющие отношение к критическим свойствам. При этом необходимо учесть возможные неблагоприятные воздействия использования системы на здоровье и безопасность человека.

Оценка требований

Данный вид деятельности состоит из решения следующей задачи. В проекте необходимо анализировать полную совокупность выявленных требований. Анализ включает в себя идентификацию и назначение приоритетов для противоречивых, пропущенных, неполных, неоднозначных, несовместимых, несоответствующих или непроверяемых требований.

Согласование требований

Данный вид деятельности состоит из решения следующих задач:

* В проекте должны решаться проблемы, относящиеся к требованиям. К ним относятся требования, которые не могут быть реализованы или которые нецелесообразно выполнять.
* В проекте должна предусматриваться обратная связь от проанализированных требований к соответствующим правообладателям для гарантии того, что их потребности и ожидания были правильно зафиксированы и выражены. Необходимо давать пояснения и достигать согласия по предложениям, касающимся противоречивых, нецелесообразных и неосуществимых требований правообладателей.
* В проекте необходимо совместно с правообладателями определять корректность выражения их требований. К этой задаче относится подтверждение того, что требования правообладателей понимаются их создателями и что разрешение противоречий в требованиях не нарушает или не компрометирует намерений правообладателей.

Регистрация требований

Данный вид деятельности состоит из решения следующих задач:

* В проекте должны регистрироваться требования правообладателей в форме, приемлемой для менеджмента требований в течение жизненного цикла и за его пределами.
* Проект должен поддерживать прослеживаемость требований правообладателей к источникам потребностей правообладателей.

1. Процесс анализа системных требований

Цель анализа системных требований состоит в преобразовании определенных требований правообладателей в совокупность необходимых системных технических требований, которыми будут руководствоваться в проекте системы.

В результате успешного осуществления анализа системных требований:

a) устанавливается определенная совокупность системных функциональных и нефункциональных требований, описывающих проблему, подлежащую решению;

b) выполняются соответствующие технические приемы оптимизации предпочитаемого проектного решения;

c) системные требования анализируются на корректность и тестируемость;

d) осмысливается воздействие системных требований на среду применения;

e) требования расставляются по приоритетам, утверждаются и обновляются;

f) устанавливается согласованность и прослеживаемость между системными требованиями и базовой линией требований заказчика;

g) оцениваются изменения базовой линии по стоимости, графикам работ и воздействию технических решений;

h) системные требования доводятся до сведения всех участвующих сторон и включаются в базовую линию.

**Виды деятельности и задачи**

В проекте должны выполняться следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса анализа системных требований:

Спецификация требований

Данный вид деятельности состоит из решения следующей задачи. Должны быть проанализированы особенности планируемого применения разрабатываемой системы для задания системных требований. Спецификация системных требований должна описывать: функции и возможности системы; требования деловой сферы, организационные и пользовательские требования; требования по безопасности, защищенности, эргономике, интерфейсам, рабочим операциям и сопровождению; проектные ограничения и квалификационные требования. Спецификация системных требований должна быть документирована.

Оценивание требований

Данный вид деятельности состоит из решения следующих задач.

Системные требования должны оцениваться на основе перечисленных ниже критериев:

a) прослеживаемость потребностей по приобретению;

b) согласованность с потребностями по приобретению;

c) тестируемость;

d) осуществимость архитектурного проекта системы;

e) осуществимость функционирования и сопровождения.

Результаты оценивания должны быть документированы.

3. Процесс проектирования архитектуры системы

Цель процесса проектирования архитектуры системы заключается в определении того, как системные требования следует распределить относительно элементов системы.

В результате успешного осуществления процесса проектирования архитектуры системы:

a) определяется архитектурный проект системы, в соответствии с которым выполняется идентификация элементов системы и удовлетворяются заданные требования;

b) устанавливаются функциональные и нефункциональные системные требования;

c) требования распределяются по элементам системы;

d) определяются внутренние и внешние интерфейсы каждого системного элемента;

e) выполняется верификация между системными требованиями и архитектурой системы;

f) требования, распределенные по системным элементам и их интерфейсам, становятся прослеживаемыми к базовой линии требований заказчика;

g) поддерживается согласованность и прослеживаемость между системными требованиями и архитектурным проектом системы

h) системные требования, конструкция, архитектурный проект системы и их взаимосвязи отражаются в базовой линии и сообщаются всем участвующим сторонам;

i) в системный проект включается человеческий фактор, эргономические знания, технические приемы, методы и средства;

j) определяются и выполняются действия по проектированию, ориентированные на человека.

**Виды деятельности и задачи**

В проекте должны выполняться следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса проектирования архитектуры системы:

Создание архитектуры

Данный вид деятельности состоит из решения следующей задачи.

Должен быть определен верхний уровень архитектуры системы. Архитектура должна идентифицировать составные части технических средств, программных средств и ручных операций. Должно гарантироваться, что все системные требования распределяются между этими составными частями. Составные части конфигурации технических средств, программных средств и ручных операций должны последовательно идентифицироваться этими составными частями. Архитектура системы и системные требования, распределенные по составным частям, должны быть документированы.

Оценивание архитектуры

Данный вид деятельности состоит из решения следующей задачи.

Архитектура системы и требования к составным частям должны быть оценены с учетом перечисленных ниже критериев:

a) прослеживаемость системных требований;

b) согласованность с системными требованиями;

c) приспособленность стандартов и методов проектирования;

d) осуществимость программных составных частей, полностью удовлетворяющих назначенным требованиям;

e) осуществимость функционирования и сопровождения.

Результаты оценок должны быть документированы.

4. Процесс квалификационного тестирования системы

Цель процесса квалификационного тестирования системы заключается в подтверждении того, что реализация каждого системного требования тестируется на соответствие и система готова к поставке.

В результате успешного осуществления процесса квалификационного тестирования системы:

a) разрабатываются критерии для оценки соответствия системным требованиям;

b) комплексированная система тестируется, используя определенные критерии;

c) документируются результаты тестирования;

d) гарантируется готовность системы для поставки.

**Виды деятельности и задачи**

При реализации проекта необходимо осуществлять следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса квалификационного тестирования системы.

Квалификационное тестирование

Данный вид деятельности состоит из решения следующих задач:

* Квалификационное тестирование системы должно проводиться в соответствии с квалификационными требованиями, установленными для системы. Должны обеспечиваться гарантии проверки выполнения каждого системного требования и готовности системы к поставке. Результаты квалификационного тестирования должны быть документированы.
* Система должна быть оценена с учетом перечисленных ниже критериев:
  + тестовое покрытие системных требований;
  + соответствие ожидаемым результатам;
  + осуществимость функционирования и сопровождения.

Результаты оценки должны быть документированы.

* Разработчик должен поддерживать проведение аудитов. Результаты аудитов должны быть документированы.
* После успешного окончания аудита (если он проводился) разработчик должен доработать и подготовить поставляемый программный продукт к инсталляции и поддержке его приемки.

5. Процесс реализации

Для каждого программного элемента системы организации следует применять процесс реализации программных средств из стандарта ISO/IEC/IEEE 12207:2008 [1] для создания программного элемента.

Целью процесса реализации программных средств является создание заданных элементов системы, выполненных в виде программных продуктов или услуг.

В ходе этого процесса происходит преобразование заданных поведенческих, интерфейсных и производственных ограничений в действия, которые создают системный элемент, выполненный в виде программного продукта или услуги, известный как "программный элемент".

Результатом процесса является создание программной составной части, удовлетворяющей как требованиям к архитектурным решениям, что подтверждается посредством верификации, так и требованиям правообладателей, что подтверждается посредством валидации.

В результате успешного осуществления процесса реализации программных средств:

a) определяется стратегия реализации;

b) определяются ограничения по технологии реализации проекта;

c) изготавливается программная составная часть;

d) программная составная часть упаковывается и хранится в соответствии с соглашением о ее поставке.

В дополнение к этим действиям процесс реализации программных средств имеет следующие процессы более низкого уровня:

- процесс анализа требований к программным средствам;

- процесс проектирования архитектуры программных средств;

- процесс детального проектирования программных средств;

- процесс конструирования программных средств;

- процесс комплексирования программных средств;

- процесс квалификационного тестирования программных средств.

**Виды деятельности и задачи**

При реализации проекта необходимо осуществлять следующие виды деятельности в соответствии с принятыми в организации политиками и процедурами в отношении процесса реализации программных средств.

Стратегия реализации программных средств

Данный вид деятельности состоит из решения следующих задач:

* Если не оговорено в контракте, разработчик должен определить или выбрать модель жизненного цикла, соответствующую области применения, размерам и сложности проекта. Модель жизненного цикла должна содержать стадии, цели и выходы каждой стадии. Виды деятельности и задачи процесса реализации программных средств должны быть выбраны и отражены в модели жизненного цикла. Эти виды деятельности и задачи могут пересекаться или взаимодействовать друг с другом, могут выполняться итеративно или рекурсивно.
* Исполнитель должен:
  + документировать результаты в соответствии с процессом менеджмента программной документации;
  + передавать результаты в процесс менеджмента конфигурации программных средств и выполнять управление изменениями в соответствии с ним;
  + документировать, решать проблемы и снимать несоответствия, найденные в программных продуктах и задачах в соответствии с процессом решения проблем в программных средствах;
  + выполнять поддержку процессов в соответствии с контрактом;
  + устанавливать базовые линии и соединять элементы конфигурации в сроки, определенные приобретающей стороной и поставщиком.
* Исполнитель должен выбирать, адаптировать и применять те стандарты, методы, инструментарий и языки программирования (если не оговорено в контракте), которые документально оформлены, являются подходящими и установлены организацией для выполнения деятельности в рамках процесса реализации программных средств и поддерживающих процессов. Реализация технологических ограничений в проекте должна определяться как часть стратегии реализации программных средств.
* Исполнитель должен разрабатывать планы проведения действий процесса реализации программных средств. Планы должны включать в себя конкретные стандарты, методы, инструментарий, действия и обязанности, связанные с разработкой и квалификацией всех требований, включая безопасность и защиту. При необходимости могут разрабатываться отдельные планы. Эти планы должны документироваться и выполняться.
* При разработке или сопровождении программных продуктов могут применяться непоставляемые элементы. Однако должно гарантироваться, что функционирование и сопровождение поставляемых программных продуктов после поставки приобретающей стороне не зависит от таких элементов; другими словами, эти элементы следует также рассматривать как поставляемые.

6. Процесс анализа требований к программным средствам

Цель процесса анализа требований к программным средствам заключается в установлении требований к программным элементам системы.

В результате успешного осуществления процесса анализа требований к программным средствам:

a) определяются требования к программным элементам системы и их интерфейсам;

b) требования к программным средствам анализируются на корректность и тестируемость;

c) осознается воздействие требований к программным средствам на среду функционирования;

d) устанавливается совместимость и прослеживаемость между требованиями к программным средствам и требованиями к системе;

e) определяются приоритеты реализации требований к программным средствам;

f) требования к программным средствам принимаются и обновляются по мере необходимости;

g) оцениваются изменения в требованиях к программным средствам по стоимости, графикам работ и техническим воздействиям;

h) требования к программным средствам воплощаются в виде базовых линий и доводятся до сведения заинтересованных сторон.

**Виды деятельности и задачи**

При реализации проекта необходимо выполнять следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса анализа требований к программным средствам.

Анализ требований к программным средствам

Для каждого программного элемента (или элемента конфигурации, если он определен) данный вид деятельности состоит из решения следующих задач:

* Исполнитель должен установить и документально оформить следующие требования к программным средствам (включая спецификации характеристик качества):
  + спецификации функциональных характеристик и возможностей, включая эксплуатационные, физические характеристики и условия окружающей среды, при которых будет применяться программная составная часть;
  + внешние интерфейсы к программной составной части;
  + квалификационные требования;
  + спецификации по безопасности, включая те спецификации, которые относятся к методам функционирования и сопровождения, влиянию окружающей среды и ущербу для персонала;
  + спецификации по защите, включая спецификации, связанные с угрозами для чувствительной информации;
  + спецификации эргономических факторов, включая спецификации, связанные с ручными операциями, взаимодействием человека с оборудованием, ограничениями по персоналу и областям, требующим концентрации внимания и чувствительным к ошибкам человека и уровню его обученности;
  + описание данных и требования к базам данных;
  + инсталляция и требования к приемке поставляемого программного продукта в местах функционирования и сопровождения;
  + требования к документации пользователя;
  + операции пользователя и требования к их выполнению;
  + пользовательские требования к сопровождению.
* Исполнитель должен оценить требования к программным средствам, учитывая критерии, перечисленные ниже. Результаты оценок должны быть документально оформлены.
  + прослеживаемость к системным требованиям и к системному проекту;
  + внешняя согласованность с системными требованиями;
  + внутренняя согласованность;
  + тестируемость;
  + осуществимость программного проекта;
  + осуществимость функционирования и сопровождения.
* Исполнитель должен проводить ревизии в соответствии с процессом ревизии программных средств.

7. Процесс проектирования архитектуры программных средств

Цель процесса проектирования архитектуры программных средств заключается в обеспечении проекта для программных средств, которые реализуются и могут быть верифицированы относительно требований.

В результате успешной реализации процесса проектирования архитектуры программных средств:

a) разрабатывается проект архитектуры программных средств и устанавливается базовая линия, описывающая программные составные части, которые будут реализовывать требования к программным средствам;

b) определяются внутренние и внешние интерфейсы каждой программной составной части;

c) устанавливаются согласованность и прослеживаемость между требованиями к программным средствам и программным проектом.

**Виды деятельности и задачи**

При реализации проекта необходимо осуществлять следующие виды деятельности в соответствии с принятыми в организации политиками и процедурами в отношении процесса проектирования архитектуры программных средств.

Проектирование архитектуры программных средств

Для каждого программного элемента (или элемента конфигурации, если он определен) данный вид деятельности состоит из решения следующих задач:

* Исполнитель должен преобразовать требования к программным составным частям в архитектуру, которая описывает верхний уровень его структуры и идентифицирует программные компоненты. Необходимо гарантировать, что все требования к программным составным частям распределяются по программным компонентам и в дальнейшем уточняются для облегчения детального проектирования. Архитектуру программной составной части необходимо документировать.
* Исполнитель должен разработать и документально оформить проект верхнего уровня для внешних интерфейсов программной составной части и интерфейсов между ней и программными компонентами.
* Исполнитель должен разработать и документально оформить проект верхнего уровня для базы данных.
* Исполнитель должен разработать и документально оформить предварительные версии пользовательской документации.
* Исполнитель должен определить и документировать требования к предварительному тестированию и график работ по комплексированию программных средств.
* Исполнитель должен оценить архитектуру программной составной части, проекты по интерфейсам и базе данных, учитывая следующие критерии:
  + прослеживаемость к требованиям программной составной части;
  + внешняя согласованность с требованиями программной составной части;
  + внутренняя согласованность между программными компонентами;
  + приспособленность методов проектирования и используемых стандартов;
  + осуществимость детального проектирования;
  + осуществимость функционирования и сопровождения.
  + Результаты оценок следует оформлять документально.
* Исполнитель должен проводить ревизии в соответствии с процессом ревизии программных средств.

8. Процесс детального проектирования программных средств

Цель процесса детального проектирования программных средств заключается в обеспечении проекта для программных средств, которые реализуются и могут быть верифицированы относительно установленных требований и архитектуры программных средств, а также существенным образом детализируются для последующего кодирования и тестирования.

В результате успешного осуществления процесса детального проектирования программных средств:

а) разрабатывается детальный проект каждого программного компонента, описывающий создаваемые программные модули;

b) определяются внешние интерфейсы каждого программного модуля

c) устанавливается совместимость и прослеживаемость между детальным проектированием, требованиями и проектированием архитектуры.

**Виды деятельности и задачи**

При реализации проекта необходимо осуществлять следующие виды деятельности в соответствии с принятыми в организации политиками и процедурами в отношении процесса детального проектирования программных средств.

Детальное проектирование программных средств

Для каждой программной составной части (или составной части конфигурации, если она определена) данный вид деятельности состоит из решения следующих задач:

* Исполнитель должен разработать детальный проект для каждого программного компонента программной составной части. Программные компоненты должны быть детализированы на более низком уровне, включающем программные блоки, которые могут быть закодированы, откомпилированы и проверены. Следует гарантировать, что все требования к программным средствам распределяются от программных компонентов к программным блокам. Детальный проект должен быть документально оформлен.
* Исполнитель должен разработать и документально оформить детальный проект для внешних интерфейсов к программным составным частям, между программными компонентами и между программными блоками. Необходимо, чтобы детальный проект для интерфейсов позволял проводить кодирование без потребности в получении дополнительной информации.
* Исполнитель должен разработать и документально оформить детальный проект базы данных.
* Исполнитель должен совершенствовать пользовательскую документацию по мере необходимости.
* Исполнитель должен определять и документировать требования к тестированию и графики работ по тестированию программных блоков. Необходимо, чтобы требования к тестированию включали в себя проведение проверок программных блоков при граничных значениях параметров, установленных в требованиях.
* Исполнитель должен обновлять требования к тестированию и графики работ по комплексированию программных средств.
* Исполнитель должен оценивать детальный проект для программных средств и требования к тестированию по следующим критериям:
  + прослеживаемость к требованиям программной составной части;
  + внешняя согласованность с архитектурным проектом;
  + внутренняя согласованность между программными компонентами и программными блоками;
  + соответствие методов проектирования и используемых стандартов;
  + осуществимость тестирования;
  + осуществимость функционирования и сопровождения.

Результаты оценки должны быть документально оформлены.

* Исполнитель должен проводить ревизии в соответствии с процессом ревизии программных средств.

9. Процесс конструирования программных средств

Цель процесса конструирования программных средств заключается в создании исполняемых программных блоков, которые должным образом отражают проектирование программных средств.

В результате успешного осуществления процесса конструирования программных средств:

a) определяются критерии верификации для всех программных блоков относительно требований;

b) изготавливаются программные блоки, определенные проектом;

c) устанавливается совместимость и прослеживаемость между программными блоками, требованиями и проектом;

d) завершается верификация программных блоков относительно требований и проекта.

**Виды деятельности и задачи**

При реализации проекта необходимо выполнять следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса конструирования программных средств.

Конструирование программных средств

Для каждой программной составной части (или составной части конфигурации, если она определена) данный вид деятельности состоит из решения следующих задач:

* Исполнитель должен разработать и документально оформить:
  + каждый программный блок и базу данных;
  + процедуры тестирования и данные для тестирования каждого программного блока и базы данных.
* Исполнитель должен тестировать каждый программный блок и базу данных, гарантируя, что они удовлетворяют требованиям. Результаты тестирования должны быть документально оформлены.
* Исполнитель должен улучшать документацию пользователя при необходимости.
* Исполнитель должен совершенствовать требования к тестированию и графики работ по комплексированию программных средств.
* Исполнитель должен оценивать программный код и результаты испытаний, учитывая следующие критерии:
  + прослеживаемость к требованиям и проекту программных элементов;
  + внешнюю согласованность с требованиями и проектом для программных составных частей;
  + внутреннюю согласованность между требованиями к блокам;
  + тестовое покрытие блоков;
  + соответствие методов кодирования и используемых стандартов;
  + осуществимость комплексирования и тестирования программных средств;
  + осуществимость функционирования и сопровождения.

Результаты оценки должны быть документально оформлены.

10. Процесс квалификационного тестирования программных средств

Цель процесса квалификационного тестирования программных средств заключается в подтверждении того, что комплектованный программный продукт удовлетворяет установленным требованиям.

В результате успешного осуществления процесса квалификационного тестирования программных средств:

a) определяются критерии для комплектованных программных средств с целью демонстрации соответствия с требованиями к программным средствам;

b) комплектованные программные средства верифицируются с использованием определенных критериев;

c) записываются результаты тестирования;

d) разрабатывается и применяется стратегия регрессии для повторного тестирования комплектованного программного средства при проведении изменений в программных составных частях.

**Виды деятельности и задачи**

При реализации проекта необходимо осуществлять следующие виды деятельности в соответствии с принятыми в организации политиками и процедурами в отношении процесса квалификационного тестирования программных средств.

Квалификационное тестирование программных средств

Для каждой программной составной части (или составной части конфигурации, если она определена) данный вид деятельности состоит из решения следующих задач:

* Исполнитель должен проводить квалификационное тестирование в соответствии с квалификационными требованиями к программному элементу. Должна обеспечиваться гарантия того, что реализация каждого требования к программным средствам тестируется на соответствие. Результаты квалификационного тестирования должны быть документально оформлены.
* Исполнитель должен обновлять пользовательскую документацию по мере необходимости.
* Исполнитель должен оценивать проект, код, тесты, результаты тестирования и пользовательскую документацию, учитывая следующие критерии:
  + тестовое покрытие требований к программной составной части;
  + соответствие с ожидаемыми результатами;
  + осуществимость системного комплексирования и тестирования, если они проводятся;
  + осуществимость функционирования и сопровождения.

Результаты оценки должны быть документально оформлены.

* Исполнитель должен поддерживать проведение аудитов в соответствии с процессом аудита программных средств. Результаты аудитов должны быть документально оформлены. Если и технические, и программные средства разрабатываются или комплексируются, то аудиты могут быть отсрочены до тех пор, пока не будет выполнено системное квалификационное тестирование.
* После успешного завершения аудитов (если они проводились) исполнитель должен обновить и подготовить поставляемый программный продукт для системного комплексирования, системного квалификационного тестирования, инсталляции программных средств или поддержки приемки программных средств.

**Раздел 2**

Методология в общем случае определяется выбором иерархии абстракций и используемыми методами декомпозиции ПО как объекта разработки при реализации принципа «разделяй и властвуй». К числу широко используемых относятся объектно-ориентированный, компонентно-ориентированный, функционально-ориентированный и др. подходы и их разновидности.

**2.1 Абстракции. Методы декомпозиции. Реализация принципа «разделяй и властвуй» (divide et impera). Нотации. Выразительные изобразительные средства (языки).[6]**

Базовая парадигма в подходе к любой задаче ясна: мы должны "разделять и властвовать". Hо главным является вопрос, каким образом мы осуществляем это разделение.

Hашей целью при декомпозиции является создание модулей, которые в свою очередь представляют собой небольшие программы, взаимодействующие друг с другом по хорошо определенным и простым правилам. Что это даст:

- разработка отдельных модулей может осуществляться различными людьми независимо друг от друга, без необходимости общения друг с другом, но при этом все эти объединенные вместе программы будут функционировать правильно.

- в процессе модификации программы появится возможность корректировать отдельные модули без необходимости исправления других.

При декомпозиции задачи мы разбиваем ее на ряд подзадач таким образом, что:

1) каждая подзадача имеет один и тот же уровень рассмотрения;

2) каждая задача может быть решена независимо;

3) полученные решения могут быть объединены вместе, позволяя решить исходную проблему.

Hо большие и плохо понимаемые задачи поддаются декомпозиции с трудом. В этом случае создание отдельных компонент, способных решить соответствующие подзадачи, не приводит к тому, что объединение этих компонент позволит решить исходную задачу. Т.е. приемлемые в отдельности решения не могут быть объединены подходящим образом, если исходная задача была разделена на части непродуманно.

Абстракция представляет собой эффективный способ декомпозиции, осуществляемый посредством изменения списка детализации. Когда мы абстрагируемся от проблемы, мы предполагаем игнорирование ряда подробностей с тем, чтобы свести задачу к более простой.

Задачи абстрагирования и последующей декомпозиции типична для процесса разработки программы:

- декомпозиция используется для разбиения программы на компоненты, которые могут быть затем объединены, позволив решить основную задачу;

- абстрагирование же предполагает продуманный выбор компонент.

Мы последовательно выполняем то один, то другой из этих процессов до тех пор, пока не сведем исходную задачу к набору подзадач, решение которых известно.

Декомпозиция и абстракция являются ключевыми моментами процесса разработки программ.

В сложных программных системах количество выделенных объектов может насчитывать сотни, их композиции не будут иметь выразительного представления, даже с учетом того, что объекты разных сценариев могут совпадать, и потребуется дополнительный анализ для их отождествления.

Основными рекомендациями для декомпозициисложной системы на компоненты или модули являются:

* четкое определение цели и возможность проверки их выполнимости;
* обязательное определение входные и выходные данных;
* задание иерархии, каждый уровень которой отвечает уровню абстракции системы и позволяет скрывать те детали, которые будут отработаны на следующих уровнях.

Такая пошаговая детализация принятия решений не только сводит решение сложной задачи к нескольким более простым, но и сосредоточиться на решении общих задач разработки компонентов отдельными членами команды с применением разных инструментальных средств, влияющих на эффективное их функционирование. При этом интерфейсы между компонентами должны быть согласованными для интеграции их в единую структуру.

Полученные совокупности объектов объединяются в подсистемы с учетом таких требований:

* каждая создаваемая подсистема должна ассоциироваться с определенными элементами продукта инженерии требований (как, например, актер, сценарий, объект и т.п.);
* необязательные функции или часто изменяемые функции выделятся в подсистемы так, чтобы каждая функция, для которой прогнозируются изменения требований, была как отдельная подсистема, связанная с одним актером (изменения вызывают актером);
* интерфейс подсистемы понятен и имеет взаимосвязи с другими подсистемами.

Результаты архитектурного проектирования представляются в нотациях, которые представлены в модели анализа требований средствами диаграмм (сущность-связь, переходов в состояния, потоков данных и действий, классов и т.п.). В указанных диаграммах задействованы объекты проекта, которые детализируют заданные требования к разработке и отображают решения, которые оказывают влияние на реализацию этих требований.

Нотации проектирования позволяют представить артефакты ПО и его структуру, а также поведение системы. Существует два типа нотаций: структурные, поведенческие и множество различных их представлений.

Структурные нотации являются графическими, они используются для представления структурных аспектов проектирования, компонентов и их взаимосвязей, элементов архитектуры и их интерфейсов. К ним относятся формальные языки спецификаций и 21 проектирования: ADL (Architecture Description Language), UML (Unified Modeling Language), ERD (Entity–Relation Diagrams), IDL (Interface Description Language), классы и объекты, компоненты и классы (CRC Cards), Use Case Driven и др. Нотации включают языки описания архитектуры и интерфейса, диаграммы классов и объектов, диаграммы сущность-связь, компонентов, развертывания, а также структурные диаграммы и схемы.

Поведенческие нотации отражают динамический аспект поведения систем и их компонентов. Таким нотациям соответствуют диаграммы: Data Flow, Decision Tables, Activity, Collaboration, Pre-Post Conditions, Sequence, таблицы принятия решений, формальные языки спецификации, языки проектирования PDL и др.

**2.2 Функциональная и объектная декомпозиция (инкапсуляция, наследование, полиморфизм и позднее связывание [6]**

Процесс декомпозиции проблемы, определяющей архитектуру системы, называют парадигмой программирования, базирующейся на двух широко распространенных моделях: функции-данные и объектная модель.

Модель функции-данные исторически появилась первой. Согласно этой модели проблема декомпозируется на последовательность функций и обрабатываемых с их помощью данных. Элементами композиции служат данные и функции над ними, их представления должны быть согласованы между собой. Если изменяются некоторые данные, то пересматриваются функции, которые их обрабатывают, и определяются пути их изменений. Т.е. внесение локальных изменений в постановку проблемы требует ревизии всех данных и функций для подтверждения того, что на них не повлияли внесенные изменения.

Объектно-ориентированный подход к разработке программных систем такого недостатка не имеет. В нем общее видение решения проблемы формирования требований осуществляется исходя из следующих постулатов:

* мир составляют объекты, которые взаимодействуют между собой;
* каждому объекту присущ определенный состав свойств или атрибутов, который определяется своим именем и значениями;
* объекты могут вступать в отношения друг с другом;
* значения атрибутов и отношения могут с течением времени изменяться;
* совокупность значений атрибутов конкретного объекта в определенный момент времени определяет его состояние;
* совокупность состояний объектов определяет состояние мира объектов;
* мир и его объекты могут находиться в разных состояниях и порождать некоторые события;
* события могут быть причиною других событий или изменений состояний.

Каждый объект может принимать участие в определенных процессах, разновидностями которых есть:

* переходы из одного состояния в другое под влиянием соответствующих событий;
* возбуждение определенных событий или посылка сообщений другим объектам;
* операции, которые могут выполнять объекты;
* возможные совокупности действий, которые задают его поведение;
* обмен сообщениями.

Объект это определенная абстракция данных и поведения. Множество экземпляров с общим набором атрибутов и поведением составляет класс объектов. Определение класса связано с известным принципом сокрытия информации, суть которого можно сформулировать так: сообщайте пользователю только то, что ему нужно. Этот принцип имеет ряд преимуществ:

* пользователь избавлен от необходимости знать лишнее;
* то, что ему не сообщили, он не испортит (защита от намеренных или случайных неправомерных действий;
* все, о чем не знает пользователь, можно изменять.

Таким образом, определение объектов в соответствии с данным принципом состоит из двух частей - видимой и невидимой. Видимая часть содержит все сведения, которые требуется для того, чтобы взаимодействовать с объектом и называется интерфейса объекта. Невидимая часть содержит подробности внутреннего устройства объекта, которые "инкапсулированы" (т.е. находятся словно бы в капсуле). Так, например, если объектом является некоторый прибор, который регистрирует показатели температуры, то к видимой его части относится операция показа значения температуры. Другим важным свойством определения объектов является наследование. Один класс объектов наследует другой, если он полностью вмещает все атрибуты и поведение наследуемого класса, но имеет еще и свои атрибуты и (или) поведение. Класс, который наследуют свойства другого, называют суперклассом, а класс, которого наследует, называют подклассом. Наследственность фиксирует общие и отличающиеся черты объектов и позволяет явно выделять компоненты проблемы, которые можно использовать в ряде случаев при построении нескольких классов–наследников. Классы могут образовывать иерархии наследников произвольной глубины, где каждый отвечает определенному уровню абстракции, являясь обобщением класса-наследника и конкретизацией класса, который наследует его самого. Например, класс "число" в качестве наследников имеет подклассы: "целые числа", "комплексные числа" и "действительные числа". Все эти подклассы наследуют операции суперкласса (сложения и вычитания), но каждый из них имеет свои особенности выполнения этих операций. При объектно–ориентированном подходе модели определяются через взаимодействие определенных объектов. В модели требований фигурируют объекты, взаимодействие которых определяет проблему, решаемую с помощью программной системы, а в других моделях (модели проекта, моделях реализации и тестирования) заданный принцип взаимодействия объектов определяет сущность решения этой проблемы (модели проекта и реализации) или проверки достоверности решения (модель тестирования).

Если удается установить соответствие между объектами указанных моделей на разных стадиях (процессах) жизненного цикла продукта, то они позволяют провести трассирование требований, т.е. проследить за последовательной трансформацией требований объектов на этих стадиях. Трассирование заключается в контроле трансформаций объектов при переходе от этапа к этапу с учетом внесения изменений во все наработанные промежуточные продукты разных стадий разработки и ее завершения.

**Раздел 3**

**3.1 Системная и программная инженерия. Описание архитектуры (ПО). Стандарт ISO/IEC/IEEE 42010:2011. Языки описания архитектуры – ADL (Architecture Description Languages) [2][6]**

Архитектура проекта – высокоуровневое представление структуры, задаваемое с помощью паттернов, компонентов и их идентификация. Описание архитектуры содержит описание логики отдельных компонентов системы, достаточное для проведения работ по кодированию, и связей между ними. Существуют и другие виды структур, основанные на проектировании образцов, шаблонов, семействе программ и их каркасов.

Паттерн – это конструктивный элемент ПО, который задает взаимодействие объектов (компонентов) проектируемой системы, определение ролей и ответственности исполнителей. Основным языком задания этого элемента является UML. Паттерн может быть: структурным, в котором определяются типовые композиции структур из объектов и классов диаграммами классов, объектов, связей и др.; поведенческим, определяющим схемы взаимодействия классов объектов и их поведение диаграммами активностей, взаимодействия, потоков управления и др.; креативным, отображающим типовые схемы распределения ролей экземпляров объектов диаграммами взаимодействия, кооперации и др.

**Описания архитектуры**

Описания архитектуры включают следующее:

* определение описания архитектуры и обзорную информацию;
* определение заинтересованных сторон системы и их интересов;
* пределение каждой точки зрения на архитектуру, используемой в описании архитектуры;
* представления архитектуры и архитектурных моделей для каждой используемой точки зрения на архитектуру;
* применимые правила связей в описании архитектуры, связи и регистрацию известных несогласованностей в требуемом содержании описаний архитектуры;
* выполненные обоснования для решений архитектуры.

Глагол “включать”, указывает на то, что в описании архитектуры информация либо присутствует, либо представлена ссылка на эту информацию.

**Определение и обзор описания архитектуры**

Описание архитектуры должно идентифицировать (определять) рассматриваемую систему и включать дополнительную информацию, как определено проектом и/или организацией. Детальное содержание идентификации и дополнительных информационных объектов должно быть задано организацией и/или проектом. Примерами идентификации и дополнительной информации а описании архитектуры являются дата выпуска и статус: авторы, рецензенты, утверждающие стороны, выпускающая организация, история изменений, резюме, область применения, контекст: глоссарий, информация контроля за версией, информация по управлению конфигурацией и ссылки. Должны быть включены результаты любых оценок архитектуры или ее описания.

**Определение заинтересованных сторон и интересов**

Описание архитектуры должно определять заинтересованные стороны системы, имеющие учитываемые интересы, важные для архитектуры рассматриваемой системы.

В описании архитектуры должны быть учтены и, если применимо, определены следующие заинтересованные стороны:

* пользователи системы
* операторы системы
* приобретающие стороны системы
* владельцы системы
* поставщики системы
* разработчики системы
* строители системы
* сопровождающие стороны системы

Описание архитектуры должно определять интересы, учитываемые как основные для архитектуры рассматриваемой системы. В описании архитектуры должны быть учтены и, если применимо, определены следующие интересы:

* цепи системы
* приемлемость архитектуры для достижения целей системы
* выполнимость конструирования и развертывания системы
* потенциальные риски и воздействия системы на ее заинтересованные стороны на всем ее жизненном цикле
* сопровождаемость и развиваемость системы.

Описание архитектуры должно связывать каждый интерес с определенными заинтересованными сторонами, имеющими такой интерес.

**Точки зрения на архитектуру**

Описание архитектуры должно включать каждую используемую точку зрения на архитектуру. Каждый интерес должен быть структурирован по крайней мере одной точкой зрения.

**Архитектурные представления**

Описание архитектуры должно включать только одно архитектурное представление для каждой используемой точки зрения на архитектуру. Каждое архитектурное представление должно придерживаться соглашений его главной точки зрения на архитектуру.

Каждое архитектурное представление должно включать:

* определение и дополнительную информацию, заданную организацией и/или проектом
* определение главной точки зрения
* архитектурные модели, которые обращаются ко всем интересам, структурируемых главной точкой зрения, и охватывают с той точки зрения систему в целом
* регистрацию любых известных источников в пределах представления относительно его главной точки зрения

**Архитектурные модели**

Архитектурное представление должно быть составлено из одной или нескольких архитектурных моделей.

Каждая архитектурная модель должна включать идентификацию версии, как это задается организацией и/или проектом. Каждая архитектурная модель должна определить свой основной вид модели и придерживаться соглашений этого вида.

Архитектурная модель может быть частью более чем одного архитектурного представления.

Распределение архитектурных моделей между представлениями архитектуры разрешает описанию архитектуры структурировать различные связанные интересы без избыточности или повторения той же самой информации во множественных представлениях и уменьшает возможности для несогласованности. Распределение архитектурных моделей также разрешает объектно-ориентированный стиль описания архитектуры:

* архитектурные модели, распределенные по архитектурному представлению, могут использоваться для выражения архитектурных перспектив
* архитектурные модели, распределенные в пределах архитектурного представления, могут использоваться для выражения архитектурных структур
* архитектурные модели могут использоваться как «контейнеры» для применения архитектурных образцов или стилей архитектуры, чтобы выражать основные схемы (например, послойные, трехъярусные, децентрализованные схемы, схема «модель — представление — контроллер») в пределах представлений архитектуры.

**Архитектурные отношения**

* Согласованность в пределах описания архитектуры

Описание архитектуры должно содержать регистрацию любых известных несогласованностей через архитектурные модели и представления. В описание архитектуры следует включать анализ согласованности архитектурных моделей и представлений.

Связи и правила связи могут быть использованы для того, чтобы выразить, осуществить регистрацию, провести в жизнь и проанализировать согласованность между моделями, представлениями и другими элементами в пределах описания архитектуры.

* Связи

Каждая связь в описании архитектуры должна быть определена и описать участие элементов описания архитектуры.

Элементы описания архитектуры могут быть любыми конструкциями (заинтересованные стороны, интересы системы, точки зрения архитектуры, архитектурные представления, виды моделей, архитектурные модели, архитектурные решения и обоснования). Дополнительные виды элементов описания архитектуры могут быть введены после того, как определены точки зрения и виды моделей.

Каждая связь в описании архитектуры должна определить любые руководящие правила связи.

* Правила связи

Описание архитектуры должно включать относящееся к нему правило связи. Для каждого определенного правила связи описание архитектуры следует зарегистрировать, если оно сохраняется, а при изменении провести регистрацию всех выявленных нарушений. Правило связи сохраняется, если соответствующая связь может быть продемонстрирована для выполнения этого правила. Правило связи нарушается, если соответствующая связь не может быть продемонстрирована для выполнения этого правила или когда не существует никакой соответствующей связи.

**Регистрация обоснования архитектуры**

Описание архитектуры должно содержать обоснование для каждой точки зрения на архитектуру в терминах заинтересованных сторон, с учетом их интересов, видов моделей, нотаций и методов.

Описание архитектуры должно включать обоснование для каждого решения, которое было рассмотрено применительно к основному решению архитектуры.

В описание архитектуры следует включать свидетельство рассмотрения альтернатив и обоснования для сделанного выбора.

Для описания архитектуры следует осуществлять регистрацию решений архитектуры, которые рассматривались применительно к основному решению архитектуры системы. Регистрация каждого архитектурного решения относительно системы не является практичной. Зарегистрированное решение и соответствующую стратегию следует применять организации и/или проекту для установления критерия выбора основных решений, которые будут зарегистрированы и поддержаны обоснованием в описании архитектуры. Рассматриваемыми критериями являются:

* решения относительно архитектурно существенных требований
* решения, требующие больших инвестиционных усилий или времени для их формирования, реализации или внедрения
* решения, воздействующие на основные заинтересованные стороны или множество заинтересованных сторон
* решения, требующие сложного или неочевидного умозаключения
* решения, которые очень чувствительны к изменениям
* решения, которые могут быть дорогостоящими к изменениям
* решения, которые формируют основу для планирования и управления проектом (например, создание структуры разделения работ, прослеживание качества прохождения решений)
* решения, которые приводят к капиталовложениям или косвенным затратам.

При регистрации решений следует учитывать следующее:

* решение является уникальным
* решение утверждается
* решение связывается с интересами системы, к которым оно имеет отношение
* для решения определяется владелец
* решение связывается с элементами описания архитектуры, воздействующими на решение
* делается обоснование, связанное с решением
* определяются ограничения и предположения, которые влияют на решение
* регистрируются альтернативы, которые были рассмотрены, и их потенциальные последствия
* регистрируются последствия решения (касающиеся других решений)
* регистрируются временные отметки, когда решение было принято, когда было одобрено и когда было изменено
* предоставляются цитаты по источникам дополнительной информации.

**Языки описания архитектуры**

Язык описания архитектуры (ADL) должен задавать:

a) определение одного или более интересов, которые будут выражены ADL;

b) определение одной или более заинтересованных сторон, имеющих эти интересы;

c) виды моделей, реализованные ADL, которые структурируют эти интересы;

d) любые точки зрения на архитектуру. ADL не должен выражать точки зрения архитектуры: это может определить один или более видов модели для использования в точках зрения архитектуры, определенных в другом месте.

e) правила связи. Cвязь видов модели согласно перечислению с).

**3.2** **Критерии «сцепления- связности» компонент архитектуры ПО по функциям и данным. Факторизация.**

Один из фундаментальных принципов структурного проектирования заключается в том, что большая система должна быть расчленена на обозримые модули. При этом существенным является то, что это расчленение должно быть выполнено таким образом, чтобы модули были как можно более независимы (критерий сцепления - coupling), и чтобы каждый модуль выполнял единственную (связанную с общей задачей) функцию (критерий связности - cohesion).

Сцепление модуля - это мера его зависимости по данным от других модулей. Характеризуется способом передачи данных. Единственным видом сцепления модулей, который рекомендуется для использования современной технологией программирования, является параметрическое сцепление (сцепление по данным по Майерсу) - данные передаются модулю либо при обращении к нему как значения его параметров, либо как результат его обращения к другому модулю для вычисления некоторой функции. Такой вид сцепления модулей реализуется на языках программирования при использовании обращений к процедурам (функциям).

Одним из способов оценки качества проекта является анализ сцепления модулей. Сцепление является мерой взаимозависимости модулей. В хорошем проекте сцепления должны быть минимизированы, т.е. модули должны быть слабозависимыми (независимыми) настолько, насколько это возможно. Слабое сцепление между модулями служит признаком хорошо спроектированной системы по следующим причинам:

* уменьшение количества соединений между двумя модулями приводит к уменьшению вероятности появления “волнового эффекта” (ошибка в одном модуле влияет на работу других модулей);
* минимизация риска появления “эффекта ряби” (внесение изменений, например, при исправлении ошибки, приводит к появлению новых ошибок), т.к. изменение влияет на минимальное количество модулей;
* при сопровождении модуля отсутствие необходимости беспокоиться о внутренних деталях других модулей;
* упрощение системы для понимания, насколько это возможно.

Слабое сцепление может быть достигнуто за счет комбинирования трех следующих способов действий:

* удаления необязательных связей;
* уменьшения количества необходимых связей;
* упрощением необходимых связей.

Специалистами предлагаются следующие практические рекомендации для ослабления сцепления модулей:

1. Создавайте минимальные по количеству параметров межмодульные связи.
2. Создавайте прямые (а не косвенные) межмодульные связи, поскольку интерфейс между двумя модулями наиболее понятен (и, следовательно, менее сложен), если человек может постигнуть его сразу без предварительной ссылки к некоторым другим информационным объектам.
3. Создавайте локализованные связи (например, значения списка параметров вычисляйте непосредственно перед вызовом модуля).
4. Создавайте явные связи. Красноречивым примером неявной связи является взаимодействие модуля А с модулем В за счет модификации области данных из В: для того, чтобы человек, сопровождающий модуль В понял, за счет чего модифицируется эта область данных, он будет должен проделать огромную работу.
5. Создавайте гибкие связи для облегчения модификаций.

На практике существуют три основных типа сцепления, использумых системными проектировщиками для связи модулей: нормальное сцепление, сцепление по общей области и сцепление по содержимому. С позиций структурного проектирования эти типы являются, соответственно, приемлемым, неприемлемым и запрещенным.

Два модуля А и В являются нормально сцепленными, если

* А вызывает В,
* В возвращает управление А,
* вся информация, передаваемая между А и В, представляется значениями параметров при вызове.

Связность модуля (cohesion) – внутренняя характеристика модуля, характеризующая меру прочности соединения функциональных и информационных объектов внутри одного модуля. Связность модуля характеризует степень его «плотности», степень зависимости его частей и направленности на решение определенной задачи. Чем выше связность модуля, тем меньше «ручек управления» на модуле и тем они проще. При проектировании модулей нужно стремиться к высокой связности, ибо чем выше связность, тем лучше спроектирован модуль.

Существует 7 типов связности:

* Функциональная связность
* Последовательная связность
* Информационная связность
* Процедурная связность
* Временная связность
* Логическая связность
* Связность по совпадению

**Функционально связный модуль** содержит объекты, предназначенные для решения одной единственной задачи. Примерами функционально связанных модулей являются модули проверки орфографии, вычисления заработной платы сотрудника, вычисления логарифма функции.

В **последовательно связном модуле** его объекты охватывают подзадачи, для которых выходные данные одной из подзадач являются входными для другой (открыть файл – прочитать запись – закрыть файл).

**Информационно связный модуль** содержит объекты, использующие одни и те же входные или выходные данные. Так, по ISBN книги, можно узнать ее название, автора и год издания. Эти три процедуры (определить название, определить автора, определить год издания) связаны между собой тем, что все они работают с одним и тем же информационным объектом – ISBN.

**Процедурно связный модуль** – это такой модуль, объекты которого включены в различные (возможно, несвязанные) подзадачи, в которых управление переходит от одной подзадачи к следующей (сделать зарядку, принять душ, позавтракать, одеться, отправится на работу). В отличие от последовательно связанного модуля, в котором осуществляется передача данных, в процедурно связанном модуле выполняется передача управления.

**Модуль с временной связностью** – это такой модуль, в котором объекты модуля привязаны к конкретному промежутку времени. Примером может являться модуль, осуществляющий инициализацию системы. Элементы данного модуля почти не связаны друг с другом за исключением того, что должны выполняться в определенное время.

**Модуль с логической связностью** – это такой модуль, объекты которого содействуют решению одной общей подзадачи, для которой эти объекты отобраны во внешнем по отношению к модулю мире. Так, например, альтернативы: поехать на автомобиле, на метро, на автобусе – являются средством достижения цели: добраться в како-то определенное место, из которых нужно выбрать одну.

**Модуль со связностью по совпадению** содержит объекты, которые слабо связаны друг с другом (сходить в кино, поужинать, посмотреть телевизор, проверить электронную почту).

В программных системах должны присутствовать модули, имеющие следующие три меры связности: *функциональная*, *последовательная* и *информационная*, так как другие типы связности являются крайне нежелательными и осложняют понимание и сопровождение системы.

**Раздел 4**

**4.1 Определение объемов интеграционного и системного тестирования** **[5]**

Одним из этапов разработки и документирования является этап тестирования ПО. Рассмотрим интеграционное и системное тестирование.

Интеграционное тестирование также выполняется в Design Development DD (фазе), когда основные компоненты собираются для сборки системы. Эти основные компоненты указаны в Architecture Design Document (ADD). Интеграционные тесты должны быть направлены на верификацию интерфейса основных компонентов. Интеграционное тестирование должно предшествовать тестированию системы (system testing) и следовать модульному тестированию (unit testing).

Интеграционное тестирование должно проверить, что все данные, передаваемые через  
интерфейс, согласуются со спецификациями структуры данных в ADD.  
Интеграционное тестирование должно подтвердить, что потоки управления, определенные в ADD

были реализованы.

Проекты, тестовые примеры, процедуры тестирования и отчеты по тестированию документируются в разделе «Интеграционное тестирование» в Software Verification and Validation Plan (SVVP/IT).

Системное тестирование - это процесс тестирования системы интегрированного программного обеспечения. Это тестирование может быть выполнено в среде разработки или целевой среде или их комбинации. Системное тестирование должно верифицировать соответствие системным целям, как указано в Software Requirements Document (SRD). Тестирование системы должны включать такие активности, как:

* Передача данных в систему, правильная обработка и вывод данных (т. е. end-to-end system tests);
* Практика приемочных испытаний (т. е. проверка того, что пользовательские требования будут выполнены);
* Стресс-тесты (т. е. измерение пределов производительности);
* Предварительная оценка надежности и поддержки;
* Проверка руководства пользователя программного обеспечения (Software User Manual).

Тенденции в возникновении дефектов следует отслеживать в системных тестах; поведение таких тенденций важно для оценки потенциальной приемлемости.  
Для большинства встроенных систем, а также систем, использующих специальные  
периферийные устройства, часто бывает полезно или необходимо создавать симуляторы для аппаратного обеспечения, с которым будет взаимодействовать поставляемая система. Такие симуляторы часто требуются из-за:

* Поздняя доступность конечного оборудования системы;
* Минимальное время тестирования с окончательным системным оборудованием;
* Желание избежать повреждения деликатного и / или дорогого оборудования.

Симуляторы обычно представляют собой отдельный проект. Требуется усилие, чтобы они были доступны вовремя и чтобы они были сертифицированы как идентичные, с точки зрения интерфейса, с целевым аппаратным обеспечением.

Проекты, тестовые примеры, процедуры тестирования и отчеты по тестированию документируются в разделе «Системное тестирование» в Software Verification and Validation Plan (SVVP/ST).

**4.2 Классификация методов тестирования при верификации и валидации критического ПО [7]**

Верификация и валидация являются видами деятельности, направленными на контроль качества программного обеспечения и обнаружение ошибок в нем. Имея общую цель, они отличаются источниками проверяемых в их ходе свойств, правил и ограничений, нарушение которых считается ошибкой. Верификация проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО. Кроме того, проверяется, что требования, проектные 6 решения, документация и код оформлены в соответствии с нормами и стандартами, принятыми в данной стране, отрасли и организации при разработке ПО, а также — что при их создании выполнялись все указанные в стандартах операции, в нужной последовательности. Обнаруживаемые при верификации ошибки и дефекты являются расхождениями или противоречиями между несколькими из перечисленных документов, между документами и реальной работой программы, между нормами стандартов и реальным процессами разработки и сопровождения ПО. При этом принятие решения о том, какой именно документ подлежит исправлению (может быть, и оба) является отдельной задачей. Валидация проверяет соответствие любых создаваемых или используемых в ходе разработки и сопровождения ПО артефактов нуждам и потребностям пользователей и заказчиков этого ПО, с учетом законов предметной области и ограничений контекста использования ПО. Эти нужды и потребности чаще всего не зафиксированы документально — при фиксации они превращаются в описание требований, один из артефактов процесса разработки ПО. Поэтому валидация является менее формализованной деятельностью, чем верификация. Она всегда проводится с участием представителей заказчиков, пользователей, бизнес-аналитиков или экспертов в предметной области — тех, чье мнение можно считать достаточно хорошим выражением реальных нужд и потребностей пользователей, заказчиков и других заинтересованных лиц. Методы ее выполнения часто используют специфические техники выявления знаний и действительных потребностей участников.   
Тестирование (testing) является методом верификации, в рамках которого результаты работы тестируемой системы или компонента в ситуациях из выделенного конечного набора проверяются на соответствие проектным решениям, требованиям, общим задачам проекта, в рамках которого эта система разрабатывается или сопровождается. Ситуации, в которых выполняется тестирование, называют тестовыми ситуациями (test situations, test purposes), а процедуры, описывающие процесс создания этих ситуаций и проверки, которые необходимо выполнить над полученными результатами, — тестами.

Тестирование, как и верификация вообще, служит для поиска ошибок или дефектов и для оценки качества ПО. Эффективность решения обеих этих задач во многом определяется тем, какой именно набор тестовых ситуаций выбран для проведения тестирования.

**Методы верификации ПО**

* Статический анализ свойств артефактов жизненного цикла ПО используется для проверки формализованных правил корректного построения этих артефактов и поиска часто встречающихся дефектов по некоторым шаблонам. Такой анализ хорошо автоматизируется и может быть практически полностью возложен на инструменты, хотя иногда необходимо вручную определить, например, принятые в проекте стандарты кодирования. Однако применим он лишь к коду или к определенным форматам представления проектных артефактов, и способен обнаруживать только ограниченный набор типов ошибок. Одной из известных проблем статического анализа является также следующая дилемма: либо используются строгие методы анализа, не допускающие пропуска ошибок (тех типов, что ищутся), но приводящие к большому количеству сообщений о возможных ошибках, которые таковыми не являются, либо точным является набор сообщений об ошибках, но возникает возможность пропустить ошибку. Инструменты автоматической верификации на основе статического анализа применяются достаточно широко, поскольку не требуют специальной подготовки и достаточно удобны в использовании. Большинство техник статической проверки корректности программ, доказавших свою эффективность на практике, рано или поздно становятся частью компиляторов или даже преобразуются в семантические правила языков программирования.
* Формальные методы верификации используют для анализа свойств ПО формальные модели требований, поведения ПО и его окружения. Анализ формальных моделей выполняется с помощью специфических техник, таких как дедуктивный анализ (theorem proving), проверка моделей (model checking) или абстрактная интерпретация (abstract interpretation). Формальные методы применимы только к тем свойствам, которые выражены формально в рамках некоторой математической модели, а также к тем артефактам, для которых можно построить адекватную формальную модель. Соответственно, для использования таких методов в проекте необходимо затратить значительные усилия на построение формальных моделей. К тому же, построить такие модели и провести их анализ могут только специалисты по формальным методам, которых не так много, и чьи услуги стоят достаточно дорого. Построение формальных моделей нельзя автоматизировать, для этого всегда необходим человек. Анализ их свойств в значительной мере может быть автоматизирован, и сейчас уже есть инструменты, способные анализировать формальные модели промышленного уровня сложности, однако чтобы эффективно пользоваться ими часто тоже требуется очень специфический набор навыков и знаний (в специфических разделах математической логики и алгебры). Тем не менее, в ряде областей, где последствия ошибки в системе могут оказаться чрезвычайно дорогими, формальные методы верификации активно используются. Они способны обнаруживать сложные ошибки, практически не выявляемые с помощью экспертиз или тестирования. Кроме того, формализация требований и проектных решений возможна только при их глубоком понимании, и поэтому вынуждает провести тщательнейший анализ этих артефактов, для чего часто необходима совместная работа специалистов по формальным методам и экспертов в предметной области. Их использование в этой области имеет более долгую историю, что привело к созданию более зрелых методик и инструментов. Это обусловлено более высокой стоимостью ошибок для аппаратного обеспечения, более однородной его структурой и более простыми примитивными элементами, более широким многократным использованием проектной информации, а также большей привычностью строгих ограничений и точных описаний для инженеров.
* Динамические методы верификации, в рамках которых анализ и оценка свойств программной системы делаются по результатам ее реальной работы или работы некоторых ее моделей и прототипов. Примерами такого рода методов являются обычное тестирование или имитационное тестирование, мониторинг, профилирование. Для применения динамических методов необходимо иметь работающую систему или хотя бы некоторые ее компоненты, или же их прототипы, поэтому нельзя использовать их на первых стадиях разработки. Зато с их помощью можно контролировать характеристики работы системы в ее реальном окружении, которые иногда невозможно аккуратно проанализировать с помощью других подходов. Динамически методы позволяют обнаруживать в ПО только ошибки, проявляющиеся при его работе, а, например, дефекты удобства сопровождения найти не помогут, однако, обнаруживаемые ими ошибки обычно считаются более серьезными. Для применения динамических методов верификации обычно требуется дополнительная подготовка — создание тестов, разработка тестовой системы, позволяющей их выполнять или системы мониторинга, позволяющей контролировать определенные характеристики поведения проверяемой системы. Но системы тестирования, профилирования или мониторинга могут быть сделаны один раз и использоваться многократно для широких классов ПО, лишь сами тесты необходимо готовить заново для каждой проверяемой системы. В то же время подготовка тестов на ранних этапах создания ПО позволяет обнаружить множество дефектов в описании требований и проектных документах — фактически, разработчики тестов вынуждены в ходе своей деятельности выполнять экспертизу артефактов, служащих основой для тестов. Создание набора тестов, позволяющих получить адекватную оценку качества сложной системы, является довольно трудоемкой задачей. Однако среди разработчиков промышленного ПО сложилось (не вполне верное) мнение, что тестирование является наименее ресурсоемким методом верификации, поэтому на практике оно используется для оценки свойств ПО очень широко. При этом чаще всего применяются не слишком надежные, но достаточно дешевые техники, такие как (нестрогое) вероятностное тестирование, при котором тестовые данные генерируются случайным образом, или же тестирование на основе простейших сценариев использования, проверяющие лишь наиболее простые ситуации.

Остановимся подробнее на model-checking verification.

Проверка моделей (model checking) используется для проверки выполнения набора свойств, записанных в виде утверждений какого-либо логико- алгебраического исчисления на исполнимой модели, моделирующей определенные проектные решения или код ПО.

Чаще всего для описания проверяемых свойств используется некоторая временная логика или µ-исчисление, а в качестве модели, свойства которой проверяются, выступает конечный автомат, состояния которого соответствуют наборам значений элементарных формул в проверяемых свойствах, обычно он называется моделью Крипке. Проверку модели выполняет специализированный инструмент, который либо подтверждает, что модель действительно обладает заданными свойствами, либо выдает сценарий ее работы, в конце которого эти свойства нарушаются, либо не может прийти к определенному вердикту, поскольку анализ модели требует слишком больших ресурсов.

Проверяемые свойства обычно разделяют на свойства безопасности (safety properties), означающие, что нечто нежелательное при любом варианте работы системы никогда не случается, и свойства живучести (liveness properties), означающие, наоборот, что что-то желательное рано или поздно произойдет. Иногда дополнительно выделяют свойства стабильности (или сохранности, persistence properties) — при любом сценарии работы системы заданное утверждение в некоторый момент становится истинным и с тех пор остается выполненным — и свойства справедливости (fairness properties) — некоторое утверждение при любом сценарии работы будет выполнено в бесконечном множестве моментов времени. Те или иные свойства справедливости, например, что планировщик операционной системы после любого заданного момента времени гарантирует каждому активному процессу получение управления, часто являются исходными предположениями, при выполнении которых нужно проверить свойства безопасности или живучести.

**4.3 Квалификационные испытания критического ПО. Цели. Задачи. Методология.** **[3][4]**

Рассмотрим этап процесса квалификационного тестирования ПО.

Информационно-управляющие системы (ИУС) являются ключевым фактором обеспечения важнейших характеристик гарантоспособности (надежности, готовности, обслуживаемости) и функциональной безопасности проектов, реализуемых в критических сферах, таких, как атомная энергетика, космическая деятельность.

Уровень критичности ИУС определяется тяжестью последствий аномального функционирования в диапазоне <материальные потери - ущерб окружающей среде - угроза здоровью и жизни людей> с учетом вероятности их наступления. В этих сферах наблюдается устойчивая тенденция роста объемов программно-реализуемых и программно-поддерживаемых критических функций ИУС. В силу этого возрастает зависимость безопасности использования ИУС непосредственно от качества критического ПО. Остаточные (не выявленные на этапах испытаний) дефекты ПО являются факторами риска аномального функционирования ИУС и возникновения аварийных ситуаций. Это обуславливает статус критического ПО, как важного элемента нормативного регулирования, определяющего гарантоспособность и функциональную безопасность ИУС в целом.

Базовой процедурой, реализуемой в контурах нормативного регулирования и разрешительной деятельности, являются квалификационные испытания ИУС. Одна из главных целей квалификации заключается в рентабельной оценке величин рисков аномального функционирования ИУС, связанных с остаточными дефектами критического ПО.

Обязательная квалификация и сертификация критического ПО ИУС является заключительной процедурой гарантирования социально допустимых уровней проектных рисков возникновения аварийных состояний из-за скрытых дефектов ПО.

Наиболее значимым для квалификации и сертификации критического ПО ИУС являются результаты выполнения фундаментальных методик: независимой верификации и валидации (НВиВ) и прогнозирования вероятности скрытых дефектов.

Независимость подразумевает эффективную реализацию принципов технологического и административного разнообразия, как средства повышения достоверности (снижения степени неопределенности) результатов оценки качества критического ПО.

Достижение реальной (а не декларируемой) независимости означает предоставление объективных доказательств разнообразия и эффективности используемых методов верификации критического ПО.

Независимая верификация должна быть доказательной. Это означает использование метрик количественной оценки степени разнообразия используемых методов, которые устанавливают степень неопределенности и достоверности итоговой оценки вероятности скрытых дефектов ПО. Именно доказательная НВиВ критического ПО и достоверный прогноз скрытых дефектов на ее основе определяют реальные возможности учета и регулирования рисков, связанных с недостаточным качеством критического ПО ИУС.

Главными задачами квалификации являются:

* прогноз вероятности скрытых дефектов;
* оценка полноты тестового покрытия;
* управление рентабельностью (в смысле, достижения требуемого уровня (величины) степени неопределенности оценок при минимальных затратах ресурсов.

Инструментальная среда (ИС) при квалификационных испытаниях критического ПО обеспечивает поддержку сценария доказательной независимой верификации (НВ) и прогнозирования скрытых дефектов. С помощью ИС реализуется высокий уровень компьютеризации, автоматизации и достоверности результатов квалификации критического ПО. ИС включает комплекс утилит поддержки методик (процедур) сценария на аналитическом, информационном и организационном уровнях. Общим контекстом реализации методологии доказательной НВ и прогнозирования скрытых дефектов являются квалификационные испытания информационно-управляющих систем (ИУС), включающие реализацию фундаментальных методик FMECA, FTA, PHA, HSIA и др., определенных современной нормативной базой (стандартами).

Концепция доказательной независимой верификации заключается в реализации следующих основных положений:

* использование методологии статического анализа исходного ПО как платформы доказательной независимой верификации и прогнозирования скрытых дефектов;
* использование model-checking подхода для диверсифицированного инварианто-ориентированного измерения и оценки качества критического ПО;
* формальная верификация критического ПО с использованием критерия сохранности (неизменности) инвариантов для всей области возможных вариантов использования, определенной ТЗ;
* экспериментальная калибровка методом посева тестовых дефектов чувствительности и степени разнообразия инварианто-ориентированных моделей исходного ПО для количественной оценки вероятности скрытых дефектов;
* использование для калибровки метода «капельной» инъекции тестовых дефектов для исключения эффекта «интерференция-мутация» на адресном поле исходного ПО.

**Выводы**

В процессе написания данной работы были изучены процессы жизненного цикла ПО, связанные с его разработкой. Этим процессам были сопоставлены их главные задачи и методы/методологии, применяющиеся для достижения этих задач.

В разделе методов и методологий были рассмотрены понятия декомпозиции и абстракции, виды декомпозиции, включая популярный объектно-ориентированный подход. Был рассмотрен вопрос архитектуры приложения, раскрыта важность понятий сцепления и связности.

В самом конце был рассмотрен важный этап тестирования и квалификационных испытаний ПО, необходимых для выявления дефектов на ранних этапах их образования.

Были использованы статьи и пособия, связанные с темой разработки критического ПО. Также были включены в изучение международные стандарты, определяющие ЖЦ программного обеспечения и прочие важные моменты управления проектами.

**Заключение**

Мы рассмотрели основные процессы разработки ПО:

* Процесс определения требований правообладателей;
* Процесс анализа системных требований;
* Процесс проектирования архитектуры системы;
* Процесс квалификационного тестирования системы;
* Процесс реализации ПО;
* Процесс анализа требований к программным средствам;
* Процесс проектирования архитектуры программных средств;
* Процесс детального проектирования программных средств;
* Процесс конструирования программных средств;
* Процесс квалификационного тестирования программных средств.

Рассмотрели методы и методологии, применяющиеся в процессе разработки ПО для построения эффективной и надежной системы с точки зрения архитектуры и надежности. Ввели понятия декомпозиции и абстракции как способов разбиения задачи на подзадачи, привели примеры функциональной и объектной декомпозиции. Были раскрыты основные подходы к архитектуре ПО. Рассмотрели основные виды тестирования – интеграционной и системное, провели черту между понятиями валидации и верификации. Выявили значимость квалификационных испытаний для критического ПО.

Методы и методологии, применяемые на этапе разработки ПО, были предоставлены в полном объеме в соответствии с современностью.

**Список литературы**

[1] ГОСТ Р ИСО/МЭК 12207-2010 Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств

[2] ГОСТ Р 57100-2016/ISO/IEC/IEEE 42010:2011 Системная и программная инженерия. Описание архитектуры

[3] Конорев Б.М. и другие. Квалификационные испытания критического ПО космических систем: целевая технология независимой верификации и прогнозирования скрытых дефектов.

[4] Конорев Б.М. и другие. Целевая технология рентабельной оценки надежности и функциональной безопасности критического программного обеспечения.

[5] ESA PSS-05-0 Issue 2, Software Engineering Standards (PSS-05-0) of the European Space Agency (ESA)

[6] Лаврищева Е.М. Методы программирования. Теория, инженерия, практика. Киев: Наукова думка, 2006. - 452с.

[7] МЕТОДЫ ВЕРИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В. В. Кулямин, Институт системного программирования Российской академии наук