Содержание

[Введение 3](#__RefHeading___Toc1960_1370944113)

[Объект изучения 6](#__RefHeading___Toc1958_1370944113)

[Реализация ПО 6](#__RefHeading___Toc1463_626472476)

[Группа процессов 6](#__RefHeading___Toc1465_626472476)

[Контекст процесса 6](#__RefHeading___Toc1467_626472476)

[Цель 6](#__RefHeading___Toc1469_626472476)

[Выходной продукт 6](#__RefHeading___Toc1471_626472476)

[Задачи 6](#__RefHeading___Toc1473_626472476)

[Анализ требований в системном контексте 6](#__RefHeading___Toc1475_626472476)

[Группа процессов 6](#__RefHeading___Toc1477_626472476)

[Контекст процесса 6](#__RefHeading___Toc1479_626472476)

[Цель 6](#__RefHeading___Toc1481_626472476)

[Выходной продукт 7](#__RefHeading___Toc1483_626472476)

[Задачи 7](#__RefHeading___Toc1485_626472476)

[Анализ требований к ПО 7](#__RefHeading___Toc1487_626472476)

[Группа процессов 7](#__RefHeading___Toc1489_626472476)

[Контекст процесса 7](#__RefHeading___Toc1491_626472476)

[Цель 7](#__RefHeading___Toc1493_626472476)

[Выходной продукт 7](#__RefHeading___Toc1495_626472476)

[Задачи 8](#__RefHeading___Toc1497_626472476)

[Проектирование архитектуры в системном и программном контекстах 8](#__RefHeading___Toc1499_626472476)

[Группа процессов 8](#__RefHeading___Toc1501_626472476)

[Контекст процесса 8](#__RefHeading___Toc1503_626472476)

[Цель 8](#__RefHeading___Toc1505_626472476)

[Выходной продукт 8](#__RefHeading___Toc1507_626472476)

[Задачи 9](#__RefHeading___Toc1509_626472476)

[Детальное проектирование и конструкторская разработка ПО 9](#__RefHeading___Toc1511_626472476)

[Группа процессов 9](#__RefHeading___Toc1513_626472476)

[Контекст процесса 9](#__RefHeading___Toc1515_626472476)

[Цель 9](#__RefHeading___Toc1517_626472476)

[Выходной продукт 9](#__RefHeading___Toc1519_626472476)

[Задачи 9](#__RefHeading___Toc1521_626472476)

[Квалификационное тестирование ПО в системном и программном контекстах 10](#__RefHeading___Toc1523_626472476)

[Группа процессов 10](#__RefHeading___Toc1525_626472476)

[Контекст процесса 10](#__RefHeading___Toc1527_626472476)

[Цель 10](#__RefHeading___Toc1529_626472476)

[Выходной продукт 10](#__RefHeading___Toc1531_626472476)

[Задачи 10](#__RefHeading___Toc1533_626472476)

[Разработка критического ПО 10](#__RefHeading___Toc1535_626472476)

[Определение предметной области 10](#__RefHeading___Toc1537_626472476)

[Абстракции. Методы декомпозиции 12](#__RefHeading___Toc1539_626472476)

[Нотации. Выразительные изобразительные средства 12](#__RefHeading___Toc1541_626472476)

[Функциональная и объектная декомпозиция 13](#__RefHeading___Toc1543_626472476)

[Системная и программная инженерия. Описание архитектуры ПО. Стандарт ISO/IEC/IEEE 42010:2011 13](#__RefHeading___Toc2601_1953214026)

[Критерии «сцепления-связности» компонент архитектуры ПО по функциям и данным. Факторизация 14](#__RefHeading___Toc2129_177774177)

[Характеристики хорошей модели реализации 14](#__RefHeading___Toc2131_177774177)

[Сцепление 15](#__RefHeading___Toc2133_177774177)

[Связность 15](#__RefHeading___Toc2136_177774177)

[Интеграционное и системное тестирование 16](#__RefHeading___Toc1793_177774177)

[Классификация методов тестирования при верификации и валидации критического ПО 17](#__RefHeading___Toc1795_177774177)

[Квалификационные испытания программных средств 18](#__RefHeading___Toc2482_177774177)

[Выводы 21](#__RefHeading___Toc2317_1407605772)

[Заключение 22](#__RefHeading___Toc2319_1407605772)

[Список литературы 23](#__RefHeading___Toc2603_1953214026)

# Введение

Разработка критического программного обеспечения является частным случаем разработки программного обеспечения в целом. Разработка критического, как и всего другого ПО, опирается на документ Software Body of Knowledge.

Основными облястями знания по SWEBOK являются:

* инженерия требований;
* проектирование ПО;
* конструирование ПО;
* тестирование ПО;
* сопровождение ПО.

**Требование к программному обеспечению** (англ. software requirement) это:

1. характеристика ПО, с помощью которой конечным пользователем ПО решается какая-либо задача или достигается определенная цель;
2. характеристика или свойство ПО, определенное контрактом на его разработку или другим документом (стандартом, спецификацией и т. п.).

Цель требований:

1. определение функций, условий и ограничений, присущих ПО;
2. спецификация данных, технического сопровождения и среды исполнения.

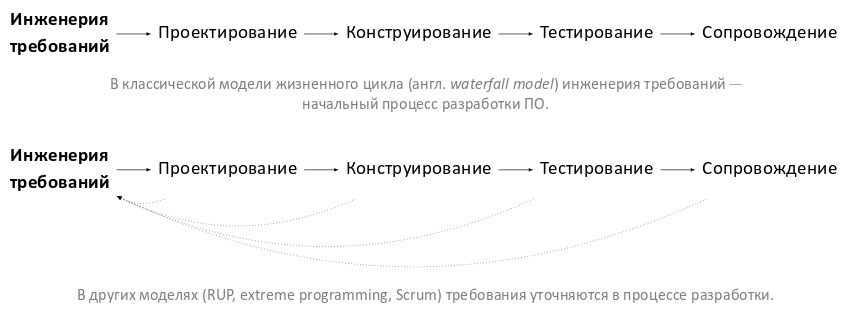
**Виды требований.** Требования к продукту и процессу — условия выполнения и режим работы ПО, ограничения на среду исполнения; определение принципов взаимодействия с другими программами.

Функциональные требования — определяют назначение и функции системы.

Нефункциональные требования — определяют условия исполнения ПО, переносимости и доступа к данным.

Системные требования - требования к программной системе в целом.

**Инженерия требований** (англ. requirements engineering) — процесс формулировки, документирования и поддержки требований к ПО, а также соответствующая область программной инженерии.



**Составляющие инженерии требований.**

*Определение*:

1. извлечение информации из договоров;
2. проведение собеседований;
3. согласование с заказчиком.

*Анализ*:

1. изучение потребностей и целей пользователей;
2. требования к системе исполнения, аппаратуре и ПО;
3. устранение конфликтов между требованиями;
4. определение приоритетов и принципов взаимодействия с окружением.

*Спецификация*:

1. формальное описание требований;
2. спецификация требований к структуре ПО,
3. функциям, качеству и документации;
4. задание архитектуры и логики системы.

*Проверка*. Проверка однозначности, непротиворечивости, полноты и реализуемости требований.

*Управление*:

1. интеграция требований во все процессы ЖЦ;
2. контроль реализации требований;
3. необходимая корректировка требований.

**Проектирование программного обеспечения** (англ. software design) — процесс определения архитектуры ПО, набора составляющих компонентов и их интерфейсов, прочих характеристик системы и конечного состава программного продукта.

Основные концепции проектирования ПО:

1. абстрагирование (отсеивание лишней информации) и уточнение (построение иерархии выполнения);
2. модульность (выделение автономных компонентов системы) и архитектура (общая структура системы, связывающая все компоненты);
3. структуризация (представления взаимоотношений между данными) и инкапусляция (отделение реализации от представления).

**Конструирование программного обеспечения** (англ. softare construction) — создание ПО из составных элементов (блоков, операторов, функций) и его проверка методами верификации и тестирования.

Техники конструирования: кодирование, верификация, модульное тестирование (unit testing), тестирование итеграции (integration testing), отладка (debugging).

Инструменты конструирования: языки конструирования; программные методы и инструментальные системы (компиляторы, СУБД, генераторы отчетов, системы управления версиями, конфигурацией, тестированием).

**Тестирование программного обеспечения** — это процесс проверки готовой программы в статике (обзоры кода, инспекции и т. п.) и динамике (прогон программы на тестовых данных) с целью обеспечить ее соответствие заданным требованиям.

**Виды тестирования:**

1. модульное (unit testing);
2. интеграционное (integration testing);
3. системное (system testing);
4. приемка (acceptance testing).

**Сопровождение программного обеспечения** (англ. software maintenance) — совокупность действий по обеспечению работы ПО, внесению изменений при выявлении ошибок, адаптации к новой среде исполнения, улучшения продуктивности или других характеристик ПО.

**Основные вопросы сопровождения ПО:**

1. технические вопросы (напр., тестирование, анализ изменений);
2. вопросы управления (напр., организация персонала);
3. экономические вопросы (оценка стоимости сопровождения);
4. измерительные вопросы (создание метрик для анализа эффективности сопровождения).

[9]

**Гарантоспособность и Безопасность критического ПО.** Целевая технология рентабельной оценки гарантоспособности и функциональной безопасности ПО критического применения основана на использовании усовершенствованной методологии статического анализа исходного ПО для независимой верификации и квалификации критического ПО. Руководящей идеей усовершенствования является измерение семантических, интервально-точностных и логических инвариантов, представляющих неизменные для всех условий использования атрибуты ПО.

Реализация подхода на процедурном уровне выглядит следующим образом [10]:

* формируется инструментированная версия исходного ПО, в которой определены контрольные точки или зонды, содержащие информацию, обеспечивающую реализацию алгебры контроля семантического, интервально-точностного и логического инвариантов ПО;
* измеряются значения инвариантов в конечных и промежуточных контрольных точках для всех реально реализованных в ПО цепочках операторных отображений в режиме рекурсивной интерпретации оценочной модели ПО;
* оцениваются траектории изменения значений семантических, интервально-точностных и логических векторов переменных для реально реализованных в ПО множеств цепочек операторных отображений;
* формируется интегральные оценки инвариантов и основанные на них метрики гарантоспособности и функциональной безопасности критического ПО;
* оценивается полнота тестового покрытия исходного ПО при измерении инвариантов, степень неопределенности измерений инвариантов и уровни рисков аномального функционирования ИУС, связанных с остаточным дефектами ПО;
* оценивается рентабельность достижения допустимых величин рисков аномального функционирования ИУС при приемлемых (минимальных) затратах ресурсов.

# Объект изучения

## Реализация ПО

### Группа процессов

Является частным случаем процесса реализации продукта со своими специфическими требованиями.

### Контекст процесса

### Цель

Цель процесса реализации программных средств заключается в создании заданных элементов системы, выполненных в виде программных продуктов или услуг.

### Выходной продукт

* определяется стратегия реализации и технологические ограничения;
* производится и упаковывается по соглашению о доставке программа.

### Задачи

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».

## Анализ требований в системном контексте

### Группа процессов

Является особым случаем общего процесса определения требований правообладателей и процесса анализа требований.

### Контекст процесса

### Цель

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.

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

### Выходной продукт

* устанавливается определенная совокупность системных функциональных и нефункциональных требований;
* устанавливаются условия использования услуг;
* выполняются соответствую щ ие технические приемы оптимизации предпочитаемого проектного решения;
* определяются ограничения для системных решений;
* достигается возможность прослеживания от требований правообладателей к правообладателям и их потребностям;
* описывается основа для определения системных требований и определяется основа для валидации соответствия услуг;
* системные требования анализирую тся на корректность и тестируемость;
* системные требования доводятся д о сведения всех участвую щ их сторон и включаю тся в базовую линию.

### Задачи

В ходе этого процесса выявляются требования правообладателей, определяются ограничения как следствие управленческих и технических решений, определяется взаимодействие с пользователем и устанавливаются требования к здоровью, безопасности и защищенности. Даннный процесс также включает в себя анализ, согласование и спецификация требований.

## Анализ требований к ПО

### Группа процессов

Является более низкоуровневым по отношению к реализации программного обеспечения.

### Контекст процесса

### Цель

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

### Выходной продукт

* определяются требования к компонентам программы и их интерфейсам;
* требования к программным средствам анализируются на корректность и тестируемость;
* устанавливается совместимость и прослеживаемость между требованиями к программным средствам и требованиями к системе;
* определяются приоритеты реализации требований к программным средствам;
* оцениваются изменения в требованиях к программным средствам по стоимости, графикам работ и техническим воздействиям.

### Задачи

В ходе этого процесса определяется верхний уровень архитектуры системы.

## Проектирование архитектуры в системном и программном контекстах

### Группа процессов

Является частным случаем процесса проектирования архитектуры и более низкоуровневым по отношению к реализации программного обеспечения.

### Контекст процесса

### Цель

Цель процесса проектирования архитектуры системы заключается в определении того, как системные требования следует распределить относительно элементов системы и в обеспечении проекта для программных средств, которые реализуются и могут быть верифицированы относительно требований.

### Выходной продукт

* определяется архитектурный проект системы;
* устанавливается базовая линия, описывающая программные составные части, которые будут реализовывать требования к программным средствам;
* устанавливаются функциональные и нефункциональные системные требования;
* требования распределяются по элементам системы;
* определяются внутренние и внешние интерфейсы каждого системного элемента;
* выполняется верификация между системными требованиями и архитектурой системы;
* устанавливаются согласованность и прослеживаемость между требованиями к программным средствам и программным проектом.

### Задачи

В ходе этого процесса исполнитель разрабатывает интерфейс для каждого компонента программы. Также долны быть разработаны и документированы внешние интерфейсы, база данных, пользовательская документация и требования к предварительному тестированию.

## Детальное проектирование и конструкторская разработка ПО

### Группа процессов

Является более низкоуровневым по отношению к реализации программного обеспечения.

### Контекст процесса

### Цель

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.

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

### Выходной продукт

* разрабатывается детальный проект каждого программного компонента, описывающий создаваемые программные модули;
* определяются внешние интерфейсы каждого программного модуля;
* устанавливается совместимость и прослеживаемость между детальным проектированием, требованиями и проектированием архитектуры;
* определяются критерии верификации для всех программных блоков относительно требований;
* изготавливаются программные блоки.

### Задачи

В ходе этого процесса производится детализация и последующая разработка компонентов, внешних интерфейсов, базы данных. Также совершенствуется документация и обновляются требования к тестированию. В ходе тестирования нужно удостовериться, что все требования удовлетворены.

## Квалификационное тестирование ПО в системном и программном контекстах

### Группа процессов

Является расширенной разновидностью процесса верификации и более низкоуровневым по отношению к реализации программного обеспечения.

### Контекст процесса

### Цель

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

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

### Выходной продукт

* разрабатываются критерии для оценки соответствия системным требованиям;
* комплексированная система тестируется;
* документируются результаты тестирования;
* разрабатывается и применяется стратегия регрессии для повторного тестирования комплексированного программного средства при проведении изменений в программных составных частях;
* гарантируется готовность системы для поставки.

### Задачи

В ходе этого процесса выполняется квалификационное тестирование в соответствии с квалификационными требованиями к каждому компоненту. Также обеспечиваются гарантии проверки выполнения каждого системного требования и готовности системы к поставке. [1]

# Разработка критического ПО

## Определение предметной области

Предметную область можно определить как объект или производственную систему со всем комплексом понятий и знаний о ее функционировании. При исследовании проблемной области необходимы знания о задачах, решаемых в производственной системе, и стоящих перед ней целях. Определяются также возможные стратегии управления и эвристические знания, используемые в процессе эксплуатации производственной системы.

Одна из первых задач, с решением которых сталкивается разработчик программной системы - это изучение, осмысление и анализ предметной области. Данная вопрос сильно влияет на все аспекты проекта: требования к системе, взаимодействие с пользователем, модель хранения данных, реализацию и т. д.

Модель предметной области должна быть документирована, храниться и поддерживаться в актуальном состоянии до этапа реализации. Для документирования могут быть использованы различные средства.

Для управления обсуждением области действия проекта можно использовать методику "будет - не будет". В простейшем случае — это список с двумя столбцами, в одном из которых записывается, что проект будет делать, а во втором - что не входит в проект. Такой список, формируется заинтересованными лицами при рассмотрении каждой бизнес-цели проекта, используя любую технику, например метод "мозгового штурма". Полученные характеристики позволяют четко определить границы проекта и довольно просто преобразуются в предположения, которые фиксируются в документе.

Бизнес-моделирование надо проводить на основе информации от экспертов предметной области, или совместно с ними. Вопросы, по сути, сводятся к "Что, почему, когда, как и кем происходит в предметной области и как оно взаимосвязано?":

1. Каковы основные понятия предметной области, их определения и взаимосвязи? Результат можно оформить в виде глоссария и/или концептуально-семантической модели предметной области.
2. На основании каких правил - международных, государственных, муниципальных, районных и т.д. законов, указов, стандартов, спецификаций, регламентов и т.д. - происходит то, что происходит в предметной области? Результат оформляется в виде структурированного списка или прикрепляется к элементам концептуальной модели.
3. Что реально (какие процессы, события, факты) происходит и в какой последовательности, взаимосвязи? Результат оформляется в виде сценариев описания бизнес-процессов или диаграмм SADT (IDEF0, IDEF3, DFD) / UML (Business Use-case Diagram + Activity Diagram + Sequence Diagram).

Какими свойствами обладает каждое из выделенных понятий - структурными и поведенческими? Результат описывается в виде таблиц с атрибутами Концептуальных сущностей или Детальной концептуальной моделью - ER — IDEF1X / UML Class Diagram.

Определение требований. Пользовательские требования нужно выявлять из общения с потенциальными пользователями системы. Вопросы:

1. На какую систему будет похожа создаваемая?
2. С какими системами и как давно вы работаете?
3. Какое у вас образование?
4. Каковы ваши ожидания от системы - что и как она должна делать, какие задачи помогать решать, как должна выглядеть?
5. Какие шаги необходимо предпринять для решения каждой задачи?
6. Критерии оценки качества системы?

Результаты анкетирования/интервьюирования обычно представляют в виде пользовательских историй (User Story, Agile) или Пользовательских сценариев (Use-case), также возможно их диаграммное представление средствами диаграмм потока работ (IDEF3), ARIS, Activity/State UML Diagram. Системные требования нужно выяснять у IT-специалистов Заказчика, если таковые имеются, из специфики контекста использования системы, опыта построения аналогичных систем (у IT-Экспертов-Архитекторов) и Специалистов по отдельным аспектам системы, значимым для данного проекта и Заказчика:

1. Будет ли система единичной или тиражируемой?
2. В каких странах она будет работать?
3. Насколько важна информация, хранящаяся, обрабатываемая и передающаяся системой?
4. Каков возможный ущерб от потери той или иной информации?
5. Сколько пользователей будет работать с системой сегодня, завтра, через год?

Переработанный результат оформляется в виде Системных требований (*Software Requirement**Specification*, стандарт IEEE-STD-830-1998). [5]

# Абстракции. Методы декомпозиции

Описания проектных решений должны содержать первичные спецификации крупных функциональных компонентов ПС, подлежащих разработке в детальном проекте создаваемой системы, и спецификаций используемых готовых компонентов, состав которых определяется при декомпозиции общей структуры системы. [2]

После этапа разработки системной структуры в процессе проектирования следует этап декомпозиции подсистем на модули. Между разбивкой системы на подсистемы и подсистем на модули нет принципиальных отличий. Однако компоненты модулей обычно меньше компонентов подсистем, поэтому можно использовать специальные модели декомпозиции. Здесь рассматриваются две модели, используемые на этапе модульной декомпозиции подсистем.

1. Объектно-ориентированная модель. Система состоит из набора взаимодействующих объектов.

2. Модель потоков данных. Система состоит из функциональных модулей, которые получают на входе данные и преобразуют их некоторым образом в выходные данные. Такой подход часто называется конвейерным.

В объектно-ориентированной модели модули представляют собой объекты с собственными состояниями и определенными операциями над этими состояниями. В модели потоков данных модули выполняют функциональные преобразования. В обеих моделях модули реализованы либо как последовательные компоненты, либо как процессы. По возможности разработчикам не стоит принимать поспешных решений о том, будет ли система параллельной или последовательной. Проектирование последовательной системы имеет ряд преимуществ: последовательные программы легче проектировать, реализовать, проверять и тестировать, чем параллельные системы, где очень сложно формализовать, управлять и проверять временные зависимости между процессами. Лучше сначала разбить систему на модули, а на этапе реализации решить, как организовать их выполнение – последовательно или параллельно. [3]

# Нотации. Выразительные изобразительные средства

Основными изобразительными средствами алгоритмов являются следующие способы их записи:

Словесный – содержание этапов вычислений задается на естественном языке в произвольной форме с требуемой детализацией.

Формульно-словесный – задание инструкций с использованием математических символов и выражений в сочетании со словесными пояснениями.

Блок-схемный – это графическое изображение логической структуры алгоритма, в котором каждый этап процесса переработки данных представляется в виде геометрических фигур (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций.

Псевдокод - позволяет формально изображать логику программы, не заботясь при этом о синтаксических особенностях конкретного языка программирования. Обычно представляет собой смесь операторов языка программирования и естественного языка. Является средством представления логики программы, которое можно применять вместо блок-схемы.

Языки программирования - изобразительные средства для непосредственной реализации программы на компьютере. Программа – алгоритм, записанный в форме, воспринимаемой компьютером. [6]

# Функциональная и объектная декомпозиция

Инкапсуляция — это механизм, который объеденяет данные и код, работающий с этими данными, а так же защищает и то, и другое от внешнего вмешательства или неправильного использования. Когда код и данные объеденены вместе, говорят, что создается так называемый «черный ящик». Обедененные таким способом код и данные называются объектом, то есть объект это то, что поддерживает инкапсуляцию.

Наследование — это процесс, посредством которого один объект может приобретать основные свойства другого и добавлять к ним черты, характерные только для него.

Полиморфизм — это свойство, которое позволяет одно и то же имя использовать для решения двух и более схожих, но технически разных задач. Применительно к ООП, целью является использование одного имени для задания общих для класса действий.

Позднее связывание относится к событиям, которые происходят в процессе выполнения программы. Вызов функции позднего связывания — это вызов, при котором адрес вызываемой функции до запуска программы неизвестен. [4]

# Системная и программная инженерия. Описание архитектуры ПО. Стандарт ISO/IEC/IEEE 42010:2011

Описания архитектуры включают следующее содержимое, как указано в остальной части этого предложения:

- идентификационная и обзорная информация об описании архитектуры;

- выявление заинтересованных сторон системы и их проблем;

- определение для каждой точки зрения архитектуры, используемой в описании архитектуры;

- представление архитектуры и модели архитектуры для каждой используемой точки зрения архитектуры;

- применимые правила соответствия описания архитектуры, соответствия описания архитектуры и запись известных несоответствий между требуемым содержимым описания архитектуры;

- обоснования принятых архитектурных решений.

Описание архитектуры должно идентифицировать заинтересованную систему и включать дополнительную информацию, определенную проектом и/или организацией.

Детальное содержание идентифицирующих и дополнительных информационных элементов должно соответствовать определению организации и/или проекта.

Описание архитектуры должно идентифицировать заинтересованные стороны системы, имеющие проблемы, которые считаются фундаментальными для архитектуры заинтересованной системы.

В описании архитектуры должны быть указаны проблемы, которые считаются фундаментальными для архитектуры системы интересов.

Описание архитектуры должно содержать ровно один вид архитектуры для каждой используемой точки зрения архитектуры. Каждая архитектура должна придерживаться конвенций своей управляющей архитектуры. Каждое представление архитектуры должно включать:

А) идентификацию и дополнительную информацию, указанную организацией и / или проектом;

B) определение его руководящей точки зрения;

C) архитектурные модели, которые учитывают все проблемы, обусловленные его руководящей точкой зрения, и охватывают всю систему с этой точки зрения;

D) запись любых известных проблем с точки зрения его руководящей точки зрения.

Представление архитектуры должно состоять из одной или нескольких моделей архитектуры.

Каждая модель архитектуры должна включать идентификацию версии, указанную организацией и / или проектом. Каждая модель архитектуры должна идентифицировать свой тип управляющей модели и придерживаться соглашений этого типа модели.

Описание архитектуры должно фиксировать любые известные несоответствия между моделями архитектуры и ее представлениями.

Описание архитектуры должно включать анализ соответствия ее моделей архитектуры и ее представлений.

Описание архитектуры должно включать обоснование для каждой точки зрения архитектуры, проблем, типов моделей, обозначений и методов. Описание архитектуры должно включать обоснование для каждого решения, которое рассматривается как ключевое архитектурное решение. Описание архитектуры должно служить доказательством рассмотрения альтернатив и обоснования выбора.

Язык описания архитектуры должен указывать:

А) идентификация одной или нескольких проблем, которые должны быть выражены ADL;

B) выявление одного или нескольких заинтересованных лиц, имеющих эти проблемы;

C) типовые виды, реализуемые ADL, которые определяют эти проблемы;

D) любые архитектурные точки зрения;

E) правила соответствия, относящиеся к его типовым видам на C). [7]

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

## Характеристики хорошей модели реализации

Один из фундаментальных принципов структурного проектирования заключается в том, что большая система должна быть расчленена на обозримые модули. При этом существенными является то, что это расчленение должно быть выполнено таким образом, чтобы:

* модули были как можно более независимыми (критерий сцепления(coupling) – при создании систем необходимо стремиться к максимальной независимости модулей, т.е. связанность модулей должна быть минимальной.)
* каждый модуль выполнял единственную функцию (критерий связности(cohesion) – при проектировании модулей нужно стремиться к высокой связности, т.к., чем выше связность, тем лучше спроектирован модуль.).

## Сцепление

Сцепление является мерой взаимозависимости модулей. В хорошем проекте сцепления должны быть минимизированы, т.е. модули должны быть слабо зависимыми настолько, насколько это возможно.

Существуют три типа нормального сцепления: сцепление по данным, сцепление по образцу, сцепление по управлению.

На практике наиболее часто используемым типом сцепления является сцепление по данным.

* *Модули сцеплены по данным*, если они взаимодействуют через передачу параметров и при этом каждый параметр является элементарным информационным объектом. В случае небольшого количества передаваемых параметров сцепление по данным обладает наилучшими характеристиками.
* *Модули сцеплены по образцу*, если один посылает другому составной информационный объект. Пример составного объекта: *Данные о клиенте*, включающий в себя поля *Название организации, Почтовый адрес, Телефон, Номер счета* и т.д.
* *Модули сцеплены по управлению*, если один посылает другому информационный объект – флаг, предназначенный для управления его внутренней логикой. Существует два вида флага: описательный (конец файла, введенные данные и т.д.) и управляющий (читать следующую запись, установить в начало).
* *Модули связаны по общей области* в том случае, если они ссылаются на одну и ту же область глобальных данных. Связанность (сцепление) по общей области является нежелательным, так как, во-первых, ошибка в модуле, использующем глобальную область, может неожиданно проявиться в любом другом модуле; во-вторых, такие программы трудны для понимания, так как программисту трудно определить какие именно данные используются конкретным модулем.
* *Модули связаны по содержимому* в том случае, если один из них ссылается внутрь другого. Это недопустимый тип сцепления, ибо полностью противоречит принципу модульности, т.е. представления модуля в виде черного ящика.

## Связность

Связность – мера прочности соединения функциональных и информационных объектов внутри одного модуля.

Выделяют следующие уровни связности:

* *функциональная* *(functional)* функционально связный модуль содержит объекты, предназначенные для выполнения единственной задачи, пример: расчет заработной платы, вычисление логарифма функции;
* *последовательная* *(sequential)* модуль имеет последовательную связность, если его объекты охватывают подзадачи, для которых выходные данные одной из подзадач служат входными данными для следующей, пример: открыть файл – прочитать запись – закрыть файл;
* *информационная* *(informational)* информационно связный модуль содержит объекты, использующие одни и те же входные или выходные данные. Так, по ISBN (*International Standard Book Number*) 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;
* *процедурная* *(procedural)* процедурно связный модуль является модулем, объекты которого включены в различные подзадачи, в которых управление переходит от каждой подзадачи к последующей, пример: последовательность утренних процедур;
* *временная* *(temporal)* временно связным модулем является модуль, объекты которого включены в подзадачи, связанные временем исполнения, пример: установившаяся последовательность действий перед сном;
* *логическая* *(logical)* модулем с логической связностью является модуль объекты которого содействуют решению общей подзадачи, для которой эти объекты отобраны во внешнем по отношению к модулю мире, пример: на чем ехать до места отдыха (поехать автомобилем, поехать поездом, поплыть на корабле, полететь самолетом);
* *случайная (coincidental)* случайно связным модулем является модуль, объекты которого соответствуют подзадачам, незначительно связанным друг с другом.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Мера связности** | **Сцепление** | **Модифицируемость** | **Понятность** | **Сопровождаемость** |
| **Функциональная** | хорошее | хорошая | хорошая | хорошая |
| **Последовательная** | хорошее | хорошая | близкая к хорошей | хорошая |
| **Информационная** | среднее | средняя | средняя | средняя |
| **Процедурная** | приемлемое | приемлемая | приемлемая | плохая |
| **Временная** | плохое | плохая | средняя | плохая |
| **Логическая** | плохое | плохая | плохая | плохая |
| **Случайная** | плохое | плохая | плохая | плохая |

# Интеграционное и системное тестирование

Интеграционное тестирование составляет объединение программного кода, соответствующего двум или большему количеству программных модулей, и тестирование полученного в результате кода. Это должно гарантировать, что вместе они работают, как требуется, до полной интеграции и тестирования кода каждого функционального компонента. Так как отдельные модули могут включать другие модули, некоторая часть интеграции и тестирования модулей, может происходить в процессе модульного тестирования. Тестовые варианты должны покрывать все требования проекта уровня функциональных компонентов ПС. После этого следует выполнять все необходимые изменения ПС, связанные с коррекцией дефектов, выявленных в процессе верификации, а также повторное тестирование в необходимом объеме и модифицировать файлы разработки ПС и другие программные продукты, основываясь на результатах интеграционного тестирования.

Тестирование функциональных компонентов в составе программных средств в процессе разработки комплексов программ и оценки полноты тестирования осуществляются преимущественно по степени выполнения требуемых функций и по характеристикам достигаемой корректности и качества функционирования ПС в целом. Значительную помощь в повышении качества сложных, критических ПС, может оказать систематизация видов тестирования и упорядоченное их проведение при разработке. Эти виды тестирования должны быть ориентированы на дифференцированное выявление определенных классов дефектов. Для каждого вида тестирования целесообразно разрабатывать методику его выполнения с указанием проверяемых компонентов, контролируемых параметров, ожидаемых и эталонных результатов. Кроме того, при заключительных испытаниях или сертификации должно проводиться интегральное тестирование при максимально широком варьировании тестов в условиях, соответствующих нормальной и форсированной эксплуатации. [2]

# Классификация методов тестирования при верификации и валидации критического ПО

Современные способы верификации можно поделить на эмпирические (те, которые используют экспертизу), формальные (которые используют математический аппарат верификации ПО) и формальные (которые проверяют работу программу с помощью запуска), уровни автоматизации делятся на ручные, автоматические и автоматизированные.

Понятие верификации программного обеспечения в одной из нотаций обозначает символьное выполнение программы или проверку кода на наличие ошибок и уязвимости способов проверки модели.

Экспертиза является самым популярным методом тестирование программ. Другими словами, это анализ ПО, проводимым экспертом, который может быть и разработчиком так же лицом, или группой лиц, привлеченных со стороны, для оценки ПО. При том что этот способ имеет высокую функциональную пригодность и способен решать огромный круг задач тестирование программного обеспечения, при этом может быть применим к любым свойствам ПО на любом этапе тестирование программ. Качество экспертизы зависит от опыта специалистов, выполняющих ее. С помощью метода экспертизы обнаруживают от 50 до 90 % ошибок и уязвимостей ПО. Такой метод помогает обнаружить фактически любые виды ошибок и считается одним из лучших способов, но только если экспертизу проводят опытные специалисты.

Формальные методы верификации — это верификация математической модели программы, а не ее исходный код. Требования к программе определяется в виде спецификации, то есть проверяется требование спецификации на модели программы.

Если сравнивать с экспертизой, то формальный метод является более выгодным для авторизации процесса верификации и создание моделей программ. Для создание математической модели всегда нужен опытный специалист. Формальные методы наделены отличительной особенностью это высокой функциональной пригодностью и точностью, если только создана адекватная формальная модель.

Изъян формального верификации — это не всегда возможность создать более адекватную математическую модель, при этом сохранить эффективность работы ее в промышленных проектах. Этот метод можно использовать к тем проверенным участкам, которые можно учесть в формальной модели.

Преимуществом проверки моделей является допустимость автоматизации процесса тестирования и строение модели. Создание формальной модели дает увидеть код программы в виде ряд логических выражений, позволяя наблюдать свойства программы, показанные в виде спецификации.

Статический анализ программы — это исследование выполняется без фактического выполнения программы. В основном исследуется определенная версия исходного кода. Динамический анализ дает анализировать все пути выполнения программы. Используется в том случае, если исследование выполняется с помощью автоматизированныхинструментов.

Существуют две самые популярные группы методов статической верификации: это методы дедуктивного исследования программ метод проверки модели. [8]

# Квалификационные испытания программных средств

Квалификационное тестирование системы и программного продукта в целом выполняется, чтобы продемонстрировать представителю заказчика, что удовлетворены все требования технического задания, и характеристики качества соответствуют условиям контракта. Оно должно покрывать все требования в спецификациях системы и подсистем, а также требования к интерфейсу с внешней средой.

Испытания должны включать тестирование на объектной вычислительной системе.

Программа испытаний является планом проведения серии экспериментов и должна разрабатываться с позиции допустимой минимизации объема тестирования в процессе проведения испытаний для проверки выполнения требований технического задания и соответствия предъявленной документации. Программа испытаний, методики их проведения и оценки результатов, разработанные совместно заказчиком и разработчиком, должны быть согласованы и утверждены. Они должны содержать уточнения и детализацию требований технического задания и спецификаций для данного ПС, а также гарантировать корректную проверку всех заданных характеристик качества. Программа испытаний должна содержать следующие четко сформулированные разделы:

* объект испытаний, его назначение и перечень основных документов, определивших его разработку;
* цель испытаний с указанием всех требований технического задания, характеристик и атрибутов качества, подлежащих проверке, и ограничений на проведение испытаний;
* Программу испытаний, содержащую проверку комплектности спроектированного ПС в соответствие с техническим заданием;
* методики испытаний, однозначно определяющие все понятия проверяемых характеристик качества, условия и сценарии тестирования, инструментальные средства, используемые для испытаний;
* методики обработки и оценки результатов тестирования по каждому разделу Программы испытаний.

План испытаний ПС должен описывать порядок квалификационного тестирования компонентов и подсистем, тестовую внешнюю среду, которая будет использоваться при тестировании, идентифицировать выполняемые тесты и указывать план-график тестовых действий. В документе также должны быть представлены план-график тестирования и матрица трассирования тестов к требованиям спецификаций на ПС или на его компоненты.

Большой объем разнородных данных, получаемых при испытаниях крупномасштабных ПС, и разнообразие возможных способов их обработки, интерпретации и оценки приводят к тому, что важнейшими факторами становятся методики обработки и оценки результатов, а также протоколы проверки по пунктам Программы испытаний. В соответствии с методиками испытаний средства автоматизации должны обеспечивать всю полноту проверок характеристик по каждому разделу методик. Результаты испытаний фиксируются в протоколах, которые обычно содержат следующие разделы:

* назначение тестирования и раздел требований технического задания, по которому проводились испытания;
* указания разделов методик в соответствии, с которыми проводились испытания, обработка и оценка результатов;
* условия и сценарии проведения тестирования и характеристики исходных данных;
* обобщенные результаты испытаний с оценкой их на соответствие требованиям технического задания и технической документации;
* описание отличий тестовой и реальной эксплуатационной сред;
* описание обнаруженных дефектов и ошибок и рекомендуемых улучшений;
* выводы о результатах испытаний и о соответствии созданного ПС или компонента определенному разделу требований технического задания и исходных спецификаций.

Завершаются квалификационные испытания предъявлением заказчику на утверждение комплекта документов, содержащих результаты комплексных испытаний версии программных средств:

* откорректированные тексты программ и данных на языке программирования и в объектном коде, полные спецификации требований на программные компоненты и ПС в целом после полного завершения тестирования и испытаний;
* Программу испытаний ПС по всем требованиям технического задания;
* комплект методик испытаний и обработки результатов по всем разделам программы испытаний;
* тесты, сценарии и генераторы тестовых данных, использованные для испытаний программных компонентов и версии ПС в целом;
* результаты и протоколы квалификационного тестирования, функциональные и конструктивные характеристики ПС в реальной внешней среде;
* отчет о подтверждении заданного качества, полные характеристики достигнутого качества функционирования, а также степени покрытия тестами спецификации требований к ПС;
* план, методики и средства автоматизации обучения заказчика и пользователей применению испытанной версии ПС;
* комплект эксплуатационной документации, описание ПС и руководство пользователя в соответствии с условиями контракта;
* технические условия на версию ПС, базу данных и эксплуатационную документацию для тиражирования и серийного производства;
* руководство по инсталляции, генерации пользовательской версии ПС и загрузке базы данных в соответствии с условиями и характеристиками внешней среды;
* отчет о технико-экономических показателях завершенного проекта версии ПС, выполнении планов и использованных ресурсах;
* акт о завершении испытаний и готовности к поставке и/или предъявлению для сертификационных испытаний версии ПС.

Несколько иначе организуются испытания коммерческих пакетов прикладных программ, создаваемых по инициативе фирмы или коллектива разработчиков для продажи широкому кругу пользователей при отсутствии конкретного заказчика. Для таких коммерческих комплексов программ принято проводить квалификационные испытания на соответствие критериям, формализованным руководителем проекта в два последовательных этапа  Альфа и Бета тестирование. Они заключаются в нормальной и форсированной (стрессовой) опытной эксплуатации конечными пользователями оформленного программного продукта в соответствии с эксплуатационной документацией и различаются количеством участвующих пользователей и уровнем их квалификации.

Для Альфа тестирования привлекаются конечные пользователи, работающие преимущественно в той же компании, но не участвовавшие непосредственно в разработке комплекса программ.

Для Бета тестирования привлекаются добровольные пользователи (потенциальные покупатели), которым бесплатно передается версия ПС для опытной эксплуатации.

Пользователи обязуются сообщать разработчикам сведения о всех выявленных дефектах и ошибках, а также вносить изменения в программы и данные или заменять версии исключительно по указаниям разработчиков. Только после успешной эксплуатации и Бета тестирования ограниченным контингентом пользователей, руководителем проекта или фирмы разработчиков может приниматься решение о передаче ПС в продажу для широкого круга пользователей. Обобщенные результаты Бета тестирования могут использоваться как основа для сертификационных испытаний.

При Альфа и Бета испытаниях принято разделять прогрессивное и регрессионное тестирование.

Под прогрессивным  понимается тестирование новых программных компонентов, для выявления дефектов и ошибок в исходных текстах программ и спецификациях. Регрессионное тестирование предназначено для контроля качества и корректности программ и данных после проведения корректировок. Необходимость и широта регрессионного тестирования определяется тем, что значительная доля изменений после Альфа и Бета тестирования в свою очередь содержат дефекты и ошибки.

Количество тестов и длительность обоих этапов тестирования определяются экспертно разработчиками или руководителем проекта в зависимости от сложности комплекса программ и интенсивности потока изменений.

Разработчик должен:

* проводить квалификационные испытания (тестирование) на соответствие квалификационным требованиям к программному объекту. При проведении испытаний должно быть обеспечено, чтобы реализация каждого установленного требования к программному объекту была проверена на соответствие. Результаты квалификационных испытаний должны быть документально оформлены.
* уточнить документацию пользователя.
* оценить проект, программный объект, проведенные испытания, результаты испытаний и документацию пользователя по следующим критериям:
* тестовое покрытие требований к программному объекту;
* соответствие ожидаемым результатам; возможность сборки и тестирования системы (при их проведении);
* возможность эксплуатации и сопровождения.
* обеспечить проведение аудиторской проверки(ок). Результаты аудиторских проверок должны быть документально оформлены. Если при реализации конкретного проекта разрабатывались или собирались как технические, так и программные средства, то проведение аудиторских проверок может быть отложено до квалификационных испытаний системы.

Аудиторские проверки должны проводиться для обеспечения того, чтобы:

* запрограммированные программные продукты отражали проектную документацию; подготовка приемки и требования к тестированию, установленные в документации, были пригодны для приемки программных продуктов;
* тестовые данные соответствовали установленным техническим требованиям;
* программные продукты были успешно протестированы и соответствовали установленным к ним требованиям;
* отчеты об испытаниях (тестировании) были правильны и расхождения между фактическими и ожидаемыми результатами были устранены;
* документация пользователя соответствовала установленным стандартам;
* работы были выполнены в соответствии с утвержденными требованиями, планами и договором; стоимости и графики проведения работ соответствовали утвержденным планам.

После успешного завершения аудиторских проверок, если они проводились, разработчик должен:

* доработать (при необходимости) и соответствующим образом подготовить поставляемый программный продукт к сборке системы, квалификационным испытаниям системы, вводу программного продукта в действие или к обеспечению приемки программного продукта;
* определить состояние конфигурации (базовую линию) проекта и программ данного программного объекта.

Оценивание качества программного продукта при квалификационных, приемо-сдаточных испытаниях проводится комиссией заказчика, в которой участвует руководитель разработки и некоторые ведущие разработчики, или аттестованная сертификационная лаборатория.

Для определения использования комплексами программ временных ресурсов ЭВМ полезно применять рекомендации стандарта ISO 14756  Измерение и оценивание производительности программных средств компьютерных вычислительных систем. Стандарт ориентирован на оценивание: прикладных программных средств, операционных систем и вычислительных комплексов, включающих все аппаратные и программные средства. Основные рекомендации сосредоточены в двух крупных разделах и четырех нормативных приложениях. Раздел 2 содержит общее описание методов измерений, а раздел 3 – детальные процедуры измерений и оценивания производительности ПС в составе информационной системы. Описание метода измерения производительности начинается с эмуляции – имитации пользователей и потоков данных из внешней среды: их случайных характеристик и процессов; функционирования терминалов; установления параметров рабочих нагрузок пользователей и вычислительных средств.

Средства для испытаний и определения характеристик сложных комплексов программ

Для обеспечения высокого качества крупных комплексов программ необходимы соответствующие проблемно-ориентированные интегрированные системы автоматизации тестирования, способные достаточно полно заменить испытания программ с реальными объектами внешней среды. При этом высокая стоимость и риск испытаний с реальными объектами почти всегда оправдывают значительные затраты на такие интегрированные систе­мы, если предстоят испытания критических ПС с высокими требова­ниями к качеству функционирования программ, с длительным жизненным циклом и множеством развивающихся версий. *Инструментальные средства автоматизации процессов тестирования и испытаний программ, должны обеспечивать*:

* ***определение тестов*** − реализацию процесса тестирования разработчиком: ввод тестовых наборов; генерацию тестовых данных; ввод ожидаемых, эталонных ре­зультатов;
* ***выполнение участка*** ***тестируемой программы*** между контроль­ными точками, для которого средство тестирования может перехватить операторский ввод (клавиатуры, мыши и т.д.) и для которого вводимые данные могут быть отредактированы и включены в последующие тестовые наборы;
* ***управление тестами и участком программы***, для которого средство тестирования может автоматически выполнять тестовые наборы;
* ***анализ и обработку тестовых результатов*** − возможность средства тестирования автоматически анализировать тестовые результаты: сравнение ожидаемых и реальных результатов; сравнение файлов; статистическую обработку результатов;
* ***анализ покрытия тестами исходного кода для обнаружения***: операторов, которые не были выполнены; процедур, которые не были вызваны; переменных, к которым не были обраще­ния;
* ***анализ производительности программы, когда она исполняет­ся***: загрузку центрального процессора; загрузку памяти; обращения к специфицированным элементам данных и/или сегментам кода; временные характеристики функционирования испытываемой программы;
* ***моделирование внешней среды*** − поддержку процесса тестирования с помощью модели имитации данных из внешних для ПС компонентов информационной системы.

При создании генераторов тестов внешней среды применяется два принципи­ально различающихся подхода, которые условно можно назвать ***ин­тегральным*** и ***дифференциальным***.

При ***интегральном*** или эмпири­ко-статистическом подходе основой является формальное описание входной и выходной информации имитируемого объекта, а также функциональной связи между данными на его входе и выходе. При этом структура объекта и процессы, реализующиеся при реальном функционировании его компонентов, не имеют зна­чения и не моделируются. Исходные данные и характеристики для построения таких генерато­ров тестов получаются в натурных экспериментах или при исследо­вании более детальных − дифференциальных моделей.

***Дифференциальные*** или имитационные модели генераторов тестов базируются на описаниях внутренних процессов функционирования компонентов объекта моделирования, его структуры и взаи­модействия составляющих. Результаты функционирования таких моде­лей определяются адекватностью знаний о компонентах и их харак­теристиках, а также об их взаимосвязях. Для этого необходимы дос­таточно подробные сведения обо всех процессах функционирования компонентов объектов внешней среды, которые в свою очередь могут потребовать еще более глубокого моделирования их составляющих.

**Программная имитация внешней среды на ЭВМ** позволяет:

* проводить длительное непрерывное генерирование имитируе­мых данных для определения характеристик качества функционирования ПС в широком диапазоне изменения условий и параметров, что зачастую невозможно при использовании реальных объектов;
* расширять диапазоны характеристик имитируемых объектов за пределы реально существующих или доступных источников данных, а также генерировать потоки информации, отражающие перспективные характеристики создаваемых информационных систем и объектов внешней среды;
* создавать тестовые данные, соответствующие критическим или опасным ситуациям функционирования объектов внешней среды, которые невозможно или рискованно реализовать при натурных экспе­риментах;
* обеспечивать высокую повторяемость имитируемых данных при заданных условиях их генерации и возможность прекращения или приостановки имитации на любых фазах моделирования внешней сре­ды.

Одними из наиболее сложных и дорогих имитаторов внешней среды, применяемых для испытаний комплексов программ, являются модели: полета космических аппаратов; диспетчерских пунктов управления воздушным движением; объектов систем противовоздушной обороны; сложных административных систем. Подобные ***моделирующие испытательные стенды (МИС)*** проблемно – ориентированы и объем программ, моделирующих в них внешнюю среду, может даже значительно превышать объемы соответствующих испытываемых ПС. Для их реализации выделяются достаточно мощные универсальные *моделирующие ЭВМ*. Кроме того, для автоматизации разработки программ могут использоваться отдельные специализированные, *технологические ЭВМ*, что в совокупности образует инструментальную базу для обеспечения всего ЖЦ сложных комплексов программ реального времени на *объектных реализующих ЭВМ*.

*Имитация конкретных тестов с реальными характеристиками*, адекватными объектам внешней среды, является основной частью типовых моделирующих стендов. В соответствии с полной но­менклатурой и реальными характеристиками объектов создаются их интегральные или дифференциальные модели. Выбор типов моделей зависит от глубины знаний об алгоритмах функционирования объек­тов, характеристиках их компонентов и обобщенных параметрах работы объекта в целом.

В схеме типового МИС можно выделить ряд базовых компонентов. Для каждого эксперимента по испытаниям ПС реального времени следует подготавливать план сценариев тестирования и ***обобщенные исходные данные***. В моделирующей ЭВМ план и обобщенные исходные данные пре­образуются в конкретные значения параметров для задания функцио­нирования каждого имитатора или реального объекта внешней среды. Эти данные вводятся и преобразуются на моделирующей ЭВМ вне ре­ального масштаба времени и подготавливают старт сеанса функцио­нирования стенда и испытываемого ПС в реальном времени. После этого начинают генерироваться тестовые данные.

***Аналоги объектов внешней среды*** используются преимущественно для генерации тестов, представляющих коррелированные логические переменные, которые трудно описать и смоделировать на ЭВМ. Кроме того, они позволяют проверить и аттестовать некоторые программные имитаторы внешней среды, которые впоследствии играют основ­ную роль при испытаниях.

***Данные с рабочих мест операторов-пользователей*** должны отра­жать реальные характеристики воздействий на тестируемое ПС с учетом особенностей и квалификации человека, которому предстоит использовать испытываемые программы в реальной системе обработки информации. На эту часть МИС кроме первичных исходных данных от моделирующей ЭВМ могут вводиться данные обработки ряда тестов испытываемой системой. В результате через человека и его харак­теристики замыкается контур обратной связи ручного и автоматизи­рованного управления объектами внешней среды. Такое же замыкание контура автоматизированного управления возможно в аналогах и имитаторах реальных объектов.

***Данные натурных экспериментов*** с объектами внешней среды мо­гут подготавливаться заранее вне сеансов испытаний ПС, например, при отладке аппаратной части системы обработки информации. Эти данные отражают характеристики и динамику функционирования объ­ектов, которые трудно или опасно подключать для непосредственно­го взаимодействия с недостаточно проверенным ПС. Кроме того, та­кие данные могут использоваться для аттестации адекватности ими­таторов некоторых объектов внешней среды. Они полезны в тех случаях, когда создание определенных условий функционирова­ния объектов внешней среды очень дорого или опасно и может быть выполнено только в исключительных случаях. Однако данные натур­ных экспериментов не всегда удается адекватно описать условиями и обобщенными характеристиками их поведения.

При тестировании в ряде случаев необходимо иметь *эталонные характеристики данных*, поступающих на испытываемое ПС. При рабо­те с реальными объектами зачастую приходится создавать специаль­ные измерительные комплексы, которые определяют, регистрируют и подготавливают к обработке все необходимые характеристики в про­цессе реального функционирования этих объектов (например, координаты движения самолетов при испытаниях систем УВД). Такие измерения проводятся при автономном функционировании объектов или при их взаимодействии с ПС в реальном времени. Результаты измерений ис­пользуются для определения характеристик качества ПС при работе с реальными объектами.

***Имитация эталонных характеристик объектов внешней среды*** служит для определения качества функционирования ПС в идеальных условиях − при отсутствии искажений исходных данных, ошибок в измерениях их параметров, сбоев и предумышленных отка­зов. Проверка при таких исходных данных позволяет оценить харак­теристики дефектов и ошибок результатов, обусловленные недоста­точным качеством ПС. Затем эталонные данные объединяются с опре­деленными калиброванными искажениями и дефектами исходных дан­ных, что обеспечивает подготовку тестов с динамическими и ста­тистическими характеристиками, максимально приближающимися к ре­альным. На промежуточных стадиях проверки с эталонными характе­ристиками помогают разделять причины дефектов результатов, зави­сящие от искажений исходных данных и от качества испытываемых программ обработки информации.

***Синхронизация и обобщение тестовых данных*** предназначены для упорядочения тестов от источников различных типов в соответствии с реальным временем их поступления на ПС и для распределения между моделирующей и объектной ЭВМ. В результате формируются потоки тестовых данных каждого реального объекта внешней среды, которые вводятся в объектную ЭВМ в соответствии с логикой функционирования системы обработки информации через соответствующие устройства сопряжения с моделирующей ЭВМ.

***Повторяемость сеансов испытаний*** при автоматической имитации тестов обеспечивается фиксированием всех исходных данных и при­менением программного формирования псевдослучайных чисел.

***Регистрация и обработка характеристик тестовых*** данных должна обеспечивать их контроль на соответствие заданным обобщенным харак­теристикам каждого объекта внешней среды и исходным данным сеан­са испытаний.

***Разрыв исполнения программ*** для анализа промежуточных данных может производиться методом вставок специальных контрольных программ при подготовке ПС к конкретному сеансу испытаний.

***Селекция результатов испытаний*** может основываться на стратегии контроля функционирования программ ***снизу вверх***, т.е. от анализа исполнения отдельных операторов программы и далее до стохастических результатов функционирования всего ПС в динамике реаль­ного времени. При этом регистрируется избыточное количество дан­ных, из которых затем отбирается минимум, необходимый для анали­за. Может использоваться стратегия ***сверху вниз***, т.е. упорядочен­ное, иерархическое выделение в первую очередь обобщенных резуль­татов функционирования программ с последующим уточнением регист­рируемых и анализируемых результатов вплоть до детального конт­роля исполнения отмеченных программных модулей и отдельных опе­раторов. В этом случае регистрируются только те данные, которые необходимы для анализа в конкретном сеансе тестирования. При обеих стратегиях необходимо иметь возможность управлять объемом и видом выделяемой и регистрируемой информации тестирования в зависимости от целей испытаний. Данные, получаемые и выделяемые в процессе испытаний качества ПС, целесообразно делить на следующие *группы*:

* данные, характеризующие исходную тестовую информацию и выходные результаты тестирования;
* маршруты исполнения программных компонентов и их операторов при некоторых фиксированных тестовых данных;
* аномальные события, сбои, отказы и данные, характеризую­щиеся отклонением результатов тестирования от эталонов за допус­тимые пределы и ограничения;
* характеристики использования различных ресурсов объектной ЭВМ.

***Регистрация промежуточных данных*** обычно соответствует некоторым достаточно завершенным этапам функционирования ПС. Вызовы регистрирующих программ должны подчиняться опреде­ленной системе контроля динамического функционирования ПС при исходной гипотезе, что *некоторые ошибки и дефекты в программах и данных могут проявиться на любой стадии тестирования*. Однако количество вызовов регистрирующих программ и контроль промежуточных резуль­татов, требующих нарушения целостности исполнения функциональных прог­рамм, следует ограничивать, учитывая допустимые расходы ресурсов времени на их реализацию. Так как основная задача регистрации при тестировании в реальном времени состоит в обнаружении и ло­кализации ошибок и причин отказов с точностью до функциональной группы программ или модуля, то более точное определение места дефекта следует переносить на тестирование в статике вне реального времени.

Так как испытания современных крупномасштабных систем обработки информации позволяют получать такое большое количест­во контрольных данных, что достаточно полный их анализ представляет трудную методическую и техническую задачу, обработка результатов должна осуществляться, ***ие­рархически и дифференцировано***. При избытке контроли­руемых величин снижается общее быстродействие имитаторов и ПС в результате затрат времени на контроль и регистрацию. Это затруд­няет анализ качества функционирования программ в реальном вре­мени. При переходе к массовым экспериментам испытаний качества, приходится значительно сокращать количество анализируемых пара­метров и по возможности представлять их в обобщенном виде. В каждом конкретном случае необходимо стремиться к компромиссу между полнотой регистрации промежуточных данных тестирования и удобством анализа обобщенных результатов.

***Обработ­ка результатов испытаний ПС реального времени*** может быть разде­лена на две достаточно автономные части: *оперативную и обобщающую*.

***Опера­тивная обработка*** результатов тестирования должна производиться по упрощенным алгоритмам с большой пропускной способностью, обеспечивающим сохранение реального масштаба времени для всего испытываемого комплекса программ. Основная часть оперативной обработки результатов связана с замыканием контура обратной связи для ими­тации динамики функционирования управляемых объектов внешней среды. Оперативно следует производить также селекцию некоторых результатов тести­рования и их предварительную обработку для значительного сокра­щения объема сохраняемых результатов.

В оперативную обработку целесообразно включать расчет части интегральных данных, позволяющих контролировать текущий процесс обработки информации испытываемым ПС. Желательно выделять, регистрировать и отображать критические значения параметров или ситуации, угрожающие надежности и безопасности функционирования ПС. Объем таких оперативно отображаемых данных должен быть максимально сокращенным и в то же время достаточным для анализа критических ситуаций, отражающихся на качестве функционирования ПС. Эти данные должны позволять спе­циалистам, ведущим испытания, фиксировать условия, при которых проявляются дефекты в функционировании программ, с учетом того, что автоматическая регистрация всегда имеет пробелы в составе фиксируемых параметров.

***Обобщающая обработка*** накопленных результатов испытаний мо­жет производиться вне реального времени после завершения одного или серии испытаний. Основная задача при этом состоит в рас­чете различных интегральных характеристик качества функциони­рования ПС. При натурных экспериментах с внешними объектами для получения эталонных данных в реальном времени используются специаль­ные измерительные комплексы. Особые трудности при этом могут встретиться в связи с необходимостью совмещать во времени результаты исполнения испытываемых программ и данных, получаемых от внешних измерительных комплексов, информация кото­рых используется как эталонная. Решение этой задачи возможно пу­тем либо жесткой синхронизации функционирования испытываемой и измерительной систем, либо использованием для них системы едино­го времени.

Зарегистрированные и обработанные результаты испытаний должны использоваться для установления соответствия по­лученных характеристик качества заданным требованиям. При выявлении их отклонения от требований технического задания заказчика, спецификаций или декларируе­мых в документации, должны разрабатываться корректировки программ для устранения несоответствия. Для этого все этапы тестирования и испытаний ПС должны быть поддержаны системой конфи­гурационного управления версиями программных компонентов и базой данных документирования тестов, результатов испытаний и выпол­ненных корректировок программ. *Средства накопления сообщений* об отказах, ошибках, предложениях на изменения, выполненных коррек­тировках и оцененных характеристиках качества версий являются основой для конфигурационного управления развитием и совершенствованием комплекса программ.

Приведенные выше требования и рекомендации по функциям и применению МИС ориентированы на создание крупномасштабных комплексов программ, их тестирование и испытания, в основном, до передачи в регулярную эксплуатацию. После приемки заказчиком или приобретения пользователями, в процессе функциониро­вания и применения ПС должно обеспечиваться их регулярное тестирование и оценка текущего качества. Для этого в составе *комплекса программ необходимы средства, обеспечивающие*:

* генерацию тестовых наборов или хранения тестов для конт­роля работоспособности, сохранности и целостности ПС при функци­онировании и применении;
* оперативный контроль и обнаружение дефектов исполнения программ и обработки данных при использовании ПС по прямому назначению;
* реализацию процедур предварительного анализа выявленных дефектов и опера­тивное восстановление вычислительного процесса, программ и дан­ных (рестарт) после обнаружения аномалий функционирования ПС;
* мониторинг, накопление и хранение данных о выявленных дефектах, сбоях и отказах в процессе исполнения программ и обработки данных.

*Средства генерации тестов и имитации внешней среды в составе комплекса программ* предназ­начены для оперативной подготовки исходных данных при проверке различных режимов функционирования в процессе применения ПС и при диагностике проявившихся дефектов. Минимальный состав средств генерации тестов должен передаваться пользователям для контроля использования ра­бочих версий ПС в реальном времени и входить в комплект поставки каждой пользовательской версии. Для размещения таких средств мониторинга и контроля качества функционирования ПС необходимы ресур­сы внешней и оперативной памяти, а также дополнительная производительность ЭВМ. Более глубокие испытания функционирования версий и локализации ошибок следует проводить на базе, комплекса средств имитации внешней среды высшего уровня (МИС) на моделирующей ЭВМ, которые используются специалистами по испытаниям и сертификации. Часть этих средств имитации может применяться как средства нижнего уровня (пользовательские) на объектной ЭВМ для диагностики и обеспечения полного повторения ситуаций, при которых пользователем могут быть обнаружены дефекты функционирования.

Автоматизированная имитация тестов и применение МИС может не только значительно повышать качество разрабатываемого ПС, но и снижать трудоемкость его создания. Даже приближенные оценки соотношения совокупных затрат на программную имитацию с затратами на подготовку тестовых данных при реальном функционировании объектов, в большинстве случа­ев показывают высокую рентабельность программных имитаторов внешней среды. В пределе эффективность применения имитаторов внешней среды приближается к отношению затрат в единицу времени на функционирование реальных объектов и на программную имитацию тестовых данных в тех же условиях.

При использовании программных моделей на ЭВМ *достоверность генерации* тестов определяется следующими факторами:

* адекватностью имитатора моделируемому объекту внешней среды или источ­нику информации;
* инструментальной точностью средств, реализующих имитатор внешней среды;
* статистической точностью процесса имитации и объемом тес­товых данных, учитываемых при статистическом обобщении результа­тов тестирования;
* точностью дискретизации имитаторами реальных непрерывных процессов в моделируемых объектах внешней среды.

Важнейшее значение для определения характеристик ПС имеет ***адекватность имитаторов***, которая зависит от степени учета второсте­пенных факторов, характеризующих функционирование реальных объ­ектов или источников информации, при создании их моделей. Точ­ность моделей на ЭВМ прежде всего определяется алгоритмами, на которых они базируются, и полнотой учета в них всех особенностей моделируемых объектов. Кроме того, на адекватность имитации вли­яет качество программирования и уровень дефектов и ошибок в программах ими­тации. Каждый, не учитываемый в имитаторе элемент или фактор моде­лируемой системы, необходимо оценивать путем сопоставления част­ных имитируемых данных с результатами аналитических исследований или с данными, полученными на реальных системах, и определять его возможное влияние на полную требуемую точность модели и ге­нерируемых тестов с учетом других составляющих, отражающихся на достоверности имитации.

Перечисленные факторы, влияющие на ***достоверность генера­ции тестов*** в МИС, взаимозависимы, и повышение достоверности имитации за счет одного из факторов при ограниченных ресурсах приводит, как правило, к снижению досто­верности вследствие влияния остальных. Поэтому важной задачей при создании имитационных моделей является достижение наибольшей суммарной достоверности имитации и определения значений характеристик качества функционирования ПС при сбалансированном влиянии каждого из факторов.

Опыт разработки крупных ПС показал, что качество методов решения задач на отдельных этапах управления может быть объективно оценено лишь в комплексе со всей цепью управления и имитации необходимой информации: внешней обстановки, характеристик входной информации с учетом ошибок, взаимодействующих и обеспечивающих систем. Это возможно за счет создания имитационно-моделирующих стендов и обеспечения взаимодействия по типовым каналам связи с опытными образцами компонентов систем и их комплексами программ. Такой подход оказался ***наиболее целесообразным, способствующим повышению эффективности опытно-конструкторских разработок для ПС реального времени.***

# Выводы

Для данного аттестационного задания были проанализированы 10 различных литературных источников. Подборка источников включает как материалы из рекомендуемой литературы, так и подобранные самоcтоятельно.

В ходе выполнения работы использовались только актуальные стандарты и методы, такие как: ISO 12297:2008, ISO/IEC/IEEE 42010:2011, а так же SWEBOK v3. Следственно, полученные результаты являются актуальными и соответствующими принятым по состоянию на сегодня стандартам и требованиям.

Последние изменения в SWEBOK и ISO 9126 свидетельствуют о тенденциях расширения и дополнения стандартов в направлении качества, надежности и эффективности.

# Заключение

В данном аттестационном задании была рассмотрена разработка критического программного обеспечения начиная от этапа планирования и заканчивая квалификационными испытаниями.

# Список литературы

1. ГОСТ Р ИСО/МЭК 12207-2010

2. Липаев В.В. Программная инженерия / Липаев В.В. – М.:ТЕИС, 2006.– 606 с.

3. Соммервилл И. Инженерия программного обеспечения / И. Соммервилл Пер. с англ. – М.: Вильямс, 2002. – 623 с.

4. Г. Шитдт Самоучитель С++ Пер. с англ – 3 изд. – СПб: БХВ-Петербург, 2005 – 668 с.

5. http://www.intuit.ru/studies/courses/574/430/lecture/9749

6. http://www.studfiles.ru/preview/3818661/page:2/

7. ISO/IEC/IEEE 42010:2011

8. Егоров В. В., Томилова Н. И., Амиров А. Ж., Касылкасова К. Н. Методы верификации программного обеспечения // Молодой ученый. — 2016. — №21. — С. 138-141.

Режим доступа: https://moluch.ru/archive/125/34536/

9. http://softandware.org.ua/wp-content/uploads/2014/11/base-areas.pdf

10. Б. М. Конорев, Ю. Г. Алексеев, В.В. Сергиенко, В.С. Харченко, Г.Н. Чертков «Целевая технология рентабельной оценки надежности и функциональной безопасности критического программного обеспечения».