Семахин Андрей Михайлович

Родился 14.03.1963 г. в г. Кургане. Окончил среднюю школу №31 г. Кургана, Курганский машиностроительный институт, аспирантуру Московского государственного технологического университета «СТАНКИН», кандидат технических наук, доцент.

Работает на кафедре программного обеспечения автоматизированных систем с 2006 г. Преподаёт дисциплины: основы программирования, языки программирования, алгоритмы и структуры данных, компьютерная графика, web-программирование, теория информации, тестирование и управление качеством ПО.

Тема научно-исследовательской работы «Моделирование информационных систем».

Автор 92 научных и учебно-методических работ. Научные труды опубликованы в США, Великобритании, Германии, Чехии, Польше и Болгарии.

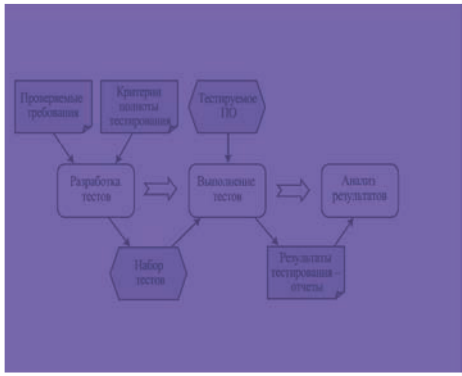
ISBN 978-5-4217-0461-4

Курганский государственный университет

редакционно-издательский центр

65-48-12

УЧЕБНОЕ ПОСОБИЕ



А.М. Семахин

МЕТОДЫ ВЕРИФИКАЦИИ И ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

Кафедра программного обеспечения автоматизированных систем

А.М. Семахин

МЕТОДЫ ВЕРИФИКАЦИИ И ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебное пособие

Курган 2018

УДК 004.41.057.2
ББК 32.973.26-018.ц.я73
С30

Рецензенты

кафедра программирования и автоматизации бизнес-процессов Шадринского государственного педагогического университета, заведующий кафедрой В.Ю. Пирогов, кандидат физико-математических наук, профессор;

кафедра математических и естественно-научных дисциплин Российской академии народного хозяйства и государственной службы при президенте Российской Федерации (Курганский филиал), заведующий кафедрой В.М. Солодовников, кандидат физико-математических наук, доцент.

Печатается по решению методического совета Курганского государственного университета.

Семахин А. М.

Методы верификации и оценки качества программного обеспечения : учебное пособие. – Курган : Изд-во КГУ, 2018. – 150 с.

В учебном пособии рассматриваются методы оценки качества программного обеспечения, описывается жизненный цикл программных средств и его процессы. Подробно рассматриваются надёжность и тестирование программного обеспечения. Приводятся сведения по стандартизации и сертификации программных средств.

Для закрепления теоретических знаний и приобретения практических навыков оценки качества программного обеспечения в учебное пособие включены вопросы и варианты заданий для самостоятельной работы.

УДК 004.41.057.2
ББК 32.973.26-018.ц.я73

ISBN 978-5-4217-0461-4

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На современном этапе развития индустриальное общество переходит к информационному. Информационные технологии базируются на аппаратном и программном обеспечении. Программное обеспечение за последнее время стало сложным и дорогостоящим. Совокупные затраты на создание, развитие и поддержку программного обеспечения превосходят затраты на аппаратное обеспечение. Соответственно сложности программ увеличивается количество ошибок в программном коде. Задача повышения качества программного обеспечения является актуальной. Для её решения используются методы верификации и тестирования программного обеспечения.

Учебное пособие рассматривает методы оценки качества программного обеспечения, жизненный цикл программных средств и его процессы. В учебном пособии описываются оценки характеристик программ на основе лексического анализа, структурной сложности программ, характеристик программ на основе процедурно-ориентированных и объектно-ориентированных метрик. Подробно представляются модели надёжности программных средств: динамические, статические и эвристические. В последних разделах учебного пособия рассматриваются теоретические положения по стандартизации и сертификации качества программного обеспечения и критериям оценок качества.

В конце учебного пособия приведены приложения, содержащие международные и государственные стандарты, регламентирующие требования, жизненный цикл, испытания и сертификацию комплексов программ, перечень оценочных элементов и показатели качества программ.

Для закрепления теоретических знаний и приобретения практических навыков в оценке качества программных средств в учебном пособии приводятся вопросы и варианты заданий самостоятельной работы.

1 Место верификации в жизненном цикле разработки программного обеспечения

1.1 Основные понятия и определения

Верификация (verification process) – процесс определения соответствия программных продуктов требованиям.

В процессе верификации проверяются условия:

- непротиворечивость требований к системе и степень учёта потребностей пользователей;
- возможность поставщика выполнить заданные требования;

- соответствие выбранных процессов жизненного цикла программных средств условиям договора;
- адекватность стандартов, процедур и среды разработки процессам жизненного цикла программных средств;
- соответствие проектных спецификаций программных средств заданным требованиям;
- корректность описания в проектных спецификациях входных и выходных данных, последовательности событий, интерфейсов и логики;
- соответствие кода проектным спецификациям и требованиям;
- тестируемость и корректность кода, соответствие принятым стандартам кодирования;
- корректность интеграции компонентов программных средств в систему;
- адекватность, полнота и непротиворечивость документации.

Аттестация (валидация) (validation process) – процесс проверки соответствия системы ожиданиям заказчика. Аттестация должна гарантировать соответствие программных средств спецификациям, требованиям и документации, возможность безопасного и надёжного применения пользователем. Аттестацию рекомендуется выполнять путем тестирования.

Тестирование (testing process) – процесс обнаружения ошибок в программном приложении.

Качество программных средств (software quality) – совокупность характеристик программного обеспечения, удовлетворяющая установленным и предполагаемым потребностям.

Факторы, влияющие на качество программного обеспечения:

- функциональные – факторы, связанные с полнотой и удобством использования реализованных функций программного средства;
- административные – факторы, связанные с квалификацией персонала, организационной структурой и управлением персоналом;
- программно-архитектурные – факторы, связанные с процессом разработки программного обеспечения, выбранными методологиями, инструментальными средствами, архитектурой программного средства.

Надежность программных средств (software reliability) – способность безотказно выполнять функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью.

Отказ программного обеспечения (software failure) – проявление ошибки в программном средстве.

Ошибка (*software error*) – сбой, некорректное поведение программы.

Примитивами надежности программного обеспечения являются завершенность, точность, автономность, устойчивость, защищенность.

Завершенность (*completeness*) – свойство, характеризующее степень обладания программного обеспечения необходимыми частями и чертами, требующимися для выполнения функций.

Точность (*accuracy*) – мера, характеризующая приемлемость величины погрешности в выдаваемых программами результатах с точки зрения предполагаемого использования.

Автономность (*self-containedness*) – свойство, характеризующее способность программного обеспечения выполнять предписанные функции без помощи или поддержки других компонент программного обеспечения.

Устойчивость (*robustness*) – свойство, характеризующее способность программного обеспечения продолжать корректное функционирование, несмотря на задание неправильных (ошибочных) входных данных.

Защищенность (*defensiveness*) – свойство, характеризующее способность программного обеспечения противостоять преднамеренным или нечаянным деструктивным (разрушающим) действиям пользователя.

Живучесть – свойство, характеризующее способность программного обеспечения продолжать корректное функционирование, несмотря на сбой частей программного обеспечения [1; 2; 3; 4; 5].

1.2 Процессы жизненного цикла программного обеспечения

Жизненный цикл программного обеспечения (software life-cycle) – совокупность этапов, связанных с изменением программного обеспечения от исходных требований до окончания эксплуатации пользователем.

Процессы жизненного цикла включают:

- организационные;
- основные;
- вспомогательные.

Схема процессов жизненного цикла программных средств приведена на рисунке 1.1.

1.2.1 Основные процессы жизненного цикла

Основные процессы жизненного цикла программных средств включают:

- *процесс приобретения (acquisition process)* – процесс, состоящий из действий и задач заказчика, приобретающего программное средство.
- *процесс поставки (supply process)* – процесс, состоящий из действий и задач, выполняемый поставщиком, который снабжает заказчика программным продуктом или услугой.

Рисунок 1.1 – Схема процессов жизненного цикла

- *процесс разработки (development process)* – процесс, состоящий из действий и задач, выполняемый разработчиком и охватывающий работы по созданию программных средств и его компонентов в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов для проверки работоспособности и качества программных продуктов, материалов для организации обучения персонала.
- *процесс эксплуатации (operation process)* – процесс, состоящий из действий и задач, выполняемый оператором и охватывающий работы по организации подготовительной работы, эксплуатационному тестированию, эксплуатации системы, поддержки пользователей.
- *процесс сопровождения (maintenance process)* – процесс, состоящий из действий и задач, выполняемый сопровождающей организацией (службой сопровождения) и охватывающий работы по модификации программного обеспечения, проверке и приёмке, снятию программного обеспечения с эксплуатации, переносу программного обеспечения в другую среду, анализу проблем и запросов на модификацию программного обеспечения [6].

1.2.2 Вспомогательные процессы жизненного цикла

Вспомогательные процессы жизненного цикла программных средств включают:

- *процесс документирования (documentation process)* – процесс, формализующий описание информации, созданной в течение жизненного цикла программных средств, включающий подготовительную работу, проектирование и разработку, выпуск документации и сопровождение.
- *процесс управления конфигурацией (configuration management process)* – процесс, использующий административные и технические процедуры на протяжении жизненного цикла программных средств для определения состояния компонентов и запросов программ, модификацию, обеспечения полноты, совместимости и корректности компонентов программ, управления хранением и поставкой программных средств.
- *процесс обеспечения качества (quality assurance process)* – процесс, обеспечивающий соответствие программных средств заданным требованиям.
- *процесс верификации (verification process)* – процесс, определяющий соответствие программного продукта требованиям спецификации.
- *процесс аттестации (validation process)* – процесс, предусматривающий определение полноты соответствия заданных требований к программному продукту.

- *процесс совместной оценки (joint review process)* – процесс, выполняющий оценку состояния работ по проекту и программным средствам.
- *процесс разрешения проблем (problem resolution process)* – процесс, выполняющий анализ и решение задач независимо от происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения.

1.2.3 Организационные процессы жизненного цикла

Организационные процессы жизненного цикла программных средств включают:

- *процесс управления (management process)* – процесс, состоящий из действий и задач, которые могут выполняться любой стороной, управляющей своими процессами.
- *процесс создания инфраструктуры (infrastructure process)* – процесс, охватывающий выбор и поддержку (сопровождение) технологии, стандартов и инструментальных средств, выбор и установку аппаратных и программных средств, используемых для разработки, эксплуатации или сопровождения программных средств.
- *процесс усовершенствования (improvement process)* – процесс, предусматривающий оценку, измерение, контроль и усовершенствование процессов жизненного цикла программных средств.
- *процесс обучения (training process)* – процесс, включающий обучение и повышение квалификации персонала.

1.3 Модели жизненного цикла программных средств

Модели жизненного цикла программных средств:

- каскадная;
- итеративная;
- спиральная.

Модель жизненного цикла программных средств определяет методологию создания программ и включает пять этапов в соответствии с ГОСТ 19.102 – 77:

1 Техническое задание:

- постановка задачи;
- выбор критериев эффективности;
- проведение предварительных научно-исследовательских работ;
- разработка технического задания.

2 Эскизный проект:

- структура входных и выходных данных;
- уточнение методов решения;
- общий алгоритм.
- разработка документации эскизного проекта.

3 Технический проект:

- уточнение структуры входных и выходных данных;
- разработка алгоритмов;
- формы данных;
- семантика и синтаксис языка;
- структура программы;
- конфигурация технических средств;
- план работ.

4 Рабочий проект:

- программирование и отладка;
- разработка документов;
- подготовка и проведение испытаний;
- корректировка программы и документов по итогам испытаний.

5 Внедрение:

- передача программы и документов для сопровождения;
- оформление акта;
- передача в Фонд алгоритмов и программ [7].

1.3.1 Каскадная модель жизненного цикла

Каскадная модель жизненного цикла основана на постепенном увеличении степени детализации описания всей разрабатываемой системы (рисунок 1.2). Особенности каскадной модели жизненного цикла программных средств:

- последовательное выполнение этапов;
- окончание предыдущего этапа до начала последующего;
- отсутствие временного перекрытия этапов;
- отсутствие или определенное ограничение возврата к предыдущим этапам;
- наличие результата только в конце разработки.

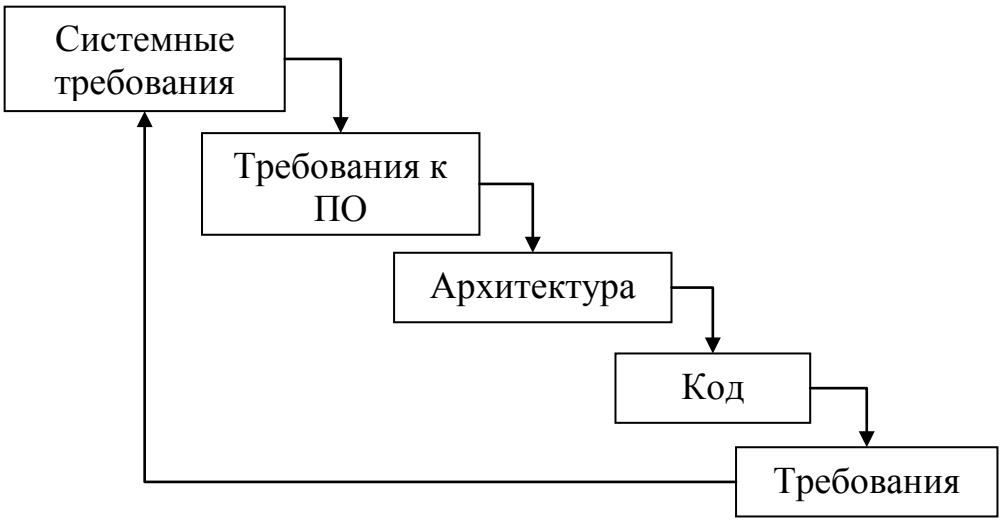


Рисунок 1.2 – Каскадная модель жизненного цикла программных средств

1.3.2 Итеративная модель жизненного цикла

Итеративная модель жизненного цикла программных средств – поэтапная модель с промежуточным контролем (рисунок 1.3). Особенность итеративной модели жизненного цикла – наличие обратных связей между этапами и возможность проверки и корректировки на каждой стадии.

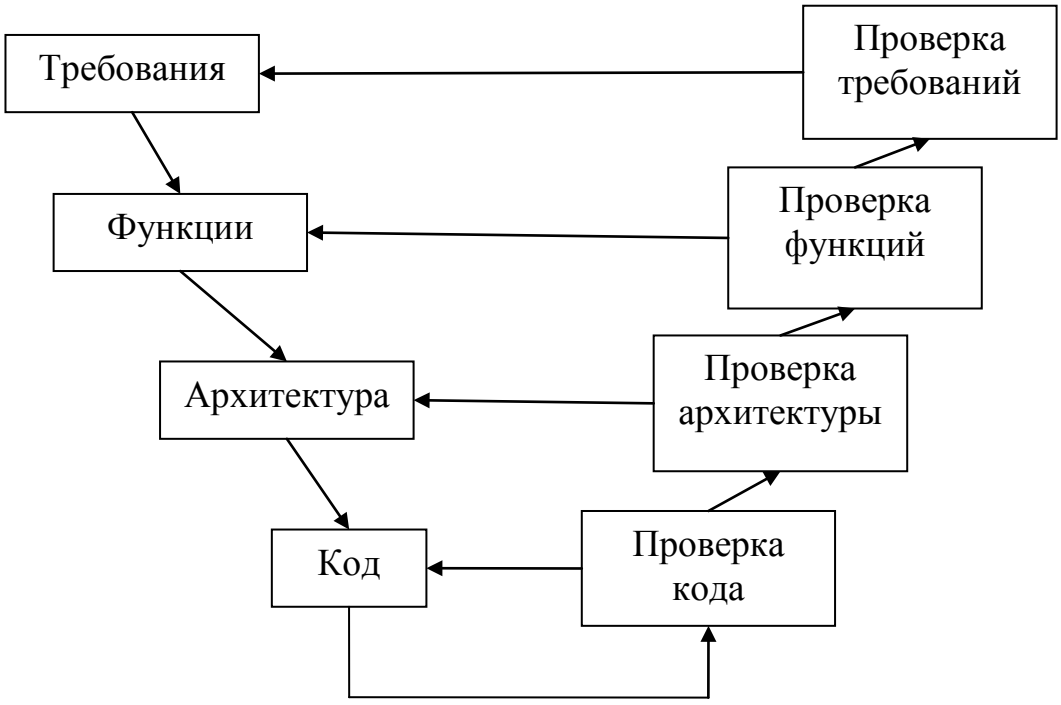


Рисунок 1.3 – Итеративная модель жизненного цикла программных средств

1.3.3 Спиральная модель жизненного цикла

Спиральная модель жизненного цикла – модель с повторяющимися этапами (рисунок 1.4). Каждый виток спирали – один каскадный или итеративный жизненный цикл. В конце каждого витка получается законченная версия системы, реализующая набор функций. Она предъявляется пользователю. На следующий виток переносится документация, разработанная на предыдущем витке, и процесс повторяется.

Система разрабатывается постепенно, проходя постоянные согласования с заказчиком. На каждом витке спирали функциональность системы расширяется, постепенно достигая полной величины [8].

Преимущества спиральной модели:

- наиболее реально (в виде эволюции) отображает разработку программного обеспечения;
- позволяет явно учитывать риск на каждом витке эволюции разработки;
- включает шаг системного подхода в итерационную структуру разработки;
- использует моделирование для уменьшения риска и совершенствования программного изделия.

Недостатки спиральной модели:

- отсутствие достаточной статистики эффективности модели;
- повышенные требования к заказчику;
- трудности контроля и управления временем разработки.

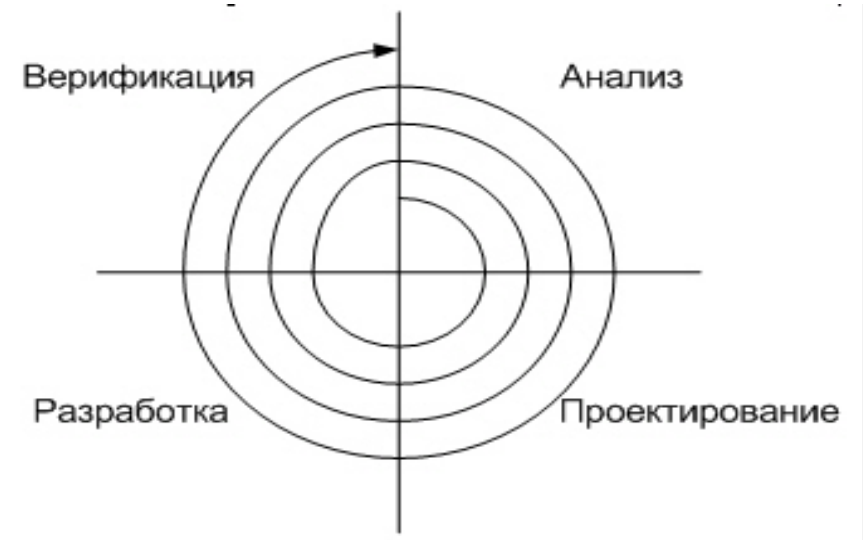


Рисунок 1.4 – Спиральная модель жизненного цикла программных средств

1.4 Классификация методов верификации программного обеспечения

Классификация методов верификации приведена на рисунке 1.5.

1.4.1 Эмпирические методы верификации программного обеспечения

Эмпирические методы используют экспертизу. Экспертиза – исследование программного обеспечения, проводимое экспертом. Экспертиза подразделяется на общую и специализированную. Общая экспертиза подразделяется на виды:

- техническая экспертиза;
- сквозной контроль;
- инспекция;
- аудит.

Техническая экспертиза – определение пригодности программного продукта для использования по назначению, соответствие спецификации и стандартам.

Сквозной контроль – анализ и оценка программы посредством проверки артефакта группой экспертов, анализирующих и делающих замечания о возможных ошибках.

Инспекция – анализ, при котором поиск ошибок и уязвимостей осуществляется в соответствии с точным планом.

Аудит – анализ программы, выполняемый специалистами, не входящими в команду проекта.

Специализированная экспертиза подразделяется на виды:

- организационная экспертиза;
- экспертиза удобства использования;
- экспертиза защищённости;
- анализ свойств архитектуры.

Организационная экспертиза – контроль руководством состояния проекта.

Экспертиза удобства использования – контроль заказчиком и пользователем удобства использования разрабатываемого программного обеспечения.

Экспертиза защищённости – контроль специалистами по информационной безопасности использования разрабатываемого программного обеспечения.

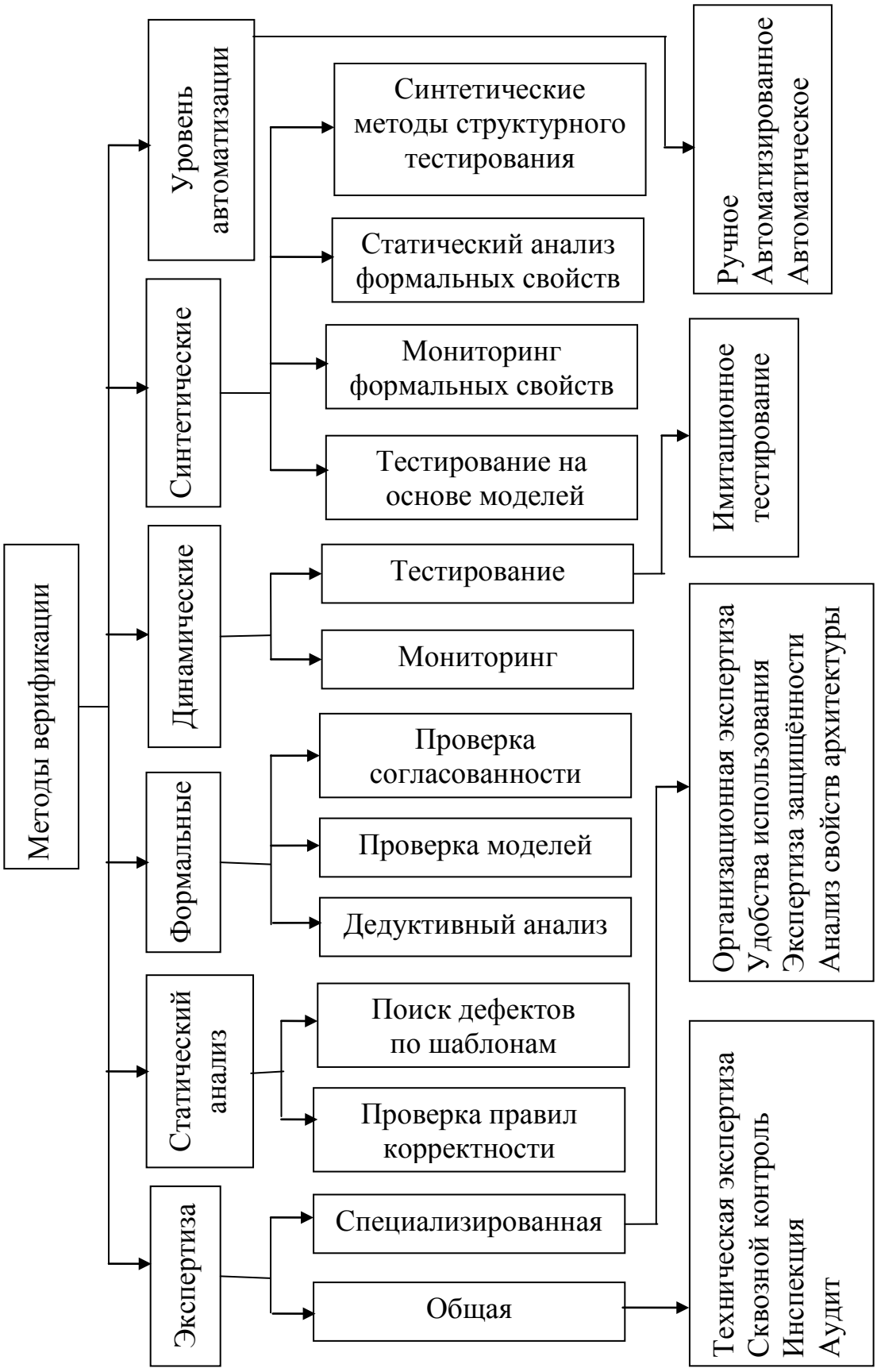


Рисунок 1.5 – Классификация методов верификации

Анализ свойств архитектуры – анализ свойств архитектуры программного обеспечения, оценка и классификация сценариев взаимодействия программного обеспечения с пользователем.

Экспертиза производится группой квалифицированных специалистов и имеет особенности:

- высокая функциональная пригодность;
- широкий круг решаемых задач верификации программного обеспечения;
- применение к любым свойствам программного обеспечения;
- использование на любом этапе разработки проекта;
- выявление любого вида ошибок.

Преимущества экспертизы:

- применимость на любом этапе проекта;
- высокая степень покрытия классов ошибок.

Недостаток экспертизы – невозможность выполнить автоматически [9].

1.4.1.1 Оценка программного обеспечения по Фагану

Оценка программного обеспечения по Фагану – оценка соответствия двух этапов жизненного цикла программного средства. Техника оценки включает 6 шагов и 4 роли участников.

Роли участников оценки программного обеспечения по Фагану:

- ведущий;
- автор;
- интерпретатор;
- оценщик.

Шаги оценки программного обеспечения по Фагану:

- планирование;
- обзор;
- подготовка;
- совместная оценка;
- доработка;
- контроль результатов [10].

1.4.2 Формальные методы верификации программного обеспечения

Формальные методы используют математическую модель программы. Требования к программной модели формируются в виде специфика-

ции. Выполнимость требований спецификации проверяется на модели программы.

Виды формальных методов:

- дедуктивный анализ;
- проверка моделей;
- проверка согласованности;
- абстрактная интерпретация.

Дедуктивный анализ – метод, используемый для доказательства соответствия программы своей спецификации, задаваемой предусловиями и постусловиями.

Проверка моделей – метод, разрабатывающий и анализирующий математическую модель с проверкой условий и ограничений.

Проверка согласованности – метод анализа соответствия между двумя исполнимыми моделями, одна из которых моделирует проверяемый проект или реальную работу системы (компонента), а вторая – проверяемые свойства. Проверяемые свойства – требования к поведению системы или ее компонента, представленные в виде обобщенного автомата (системы переходов, сети Петри), все сценарии работы которого объявляются правильными. В этом случае проверяется, что сценарии поведения реализации возможны в спецификации. Иногда устанавливается их эквивалентность, т.е. дополнительно проверяется, что все сценарии поведения спецификации есть и у реализации. Большинство методов и инструментов проверки согласованности используют для этого тестирование [10].

Абстрактная интерпретация – метод, осуществляющий верификацию динамических свойств и отсутствия ошибок при выполнении программы. Состояние программы – множество значений, которое принимает переменная.

Преимущества формальных методов:

- автоматизация процессов верификации и построения моделей программы;
- высокая точность и функциональная пригодность;

Недостатки формальных методов: ограниченный круг решаемых задач верификации программного обеспечения [9].

1.4.3 Методы статического анализа верификации программного обеспечения

Статический анализ программы – анализ, выполняющийся без запуска программы. Статический анализ позволяет проанализировать пути выполнения программы. Статический анализ хорошо автоматизируется, но способен обнаруживать ограниченный набор типов ошибок.

Инструменты автоматической верификации на основе статического анализа широко применяются на практике, поскольку не требуют специальной подготовки и удобны в использовании [10].

Методы статического анализа подразделяются на два вида:

- проверка правил корректности;
- поиск дефектов по шаблону.

1.4.4 Динамические методы верификации программного обеспечения

Динамический анализ программы – анализ, выполняющийся с запуском программы. Динамические методы включают виды:

- тестирование;
- имитационное тестирование;
- мониторинг;
- профилирование.

Мониторинг – наблюдение, запись и оценка результатов работы программного обеспечения.

Профилирование – частный случай мониторинга, включающий контроль операций с памятью и взаимодействие параллельных потоков и процессов в системе.

Тестирование – процесс обнаружения ошибок в программном коде.

Динамические методы обнаруживают ошибки в программном обеспечении, проявляющиеся при работе программного средства. Для их применения необходимо разработать набор тестов и тестовую систему их выполнения.

1.4.5 Синтетические методы верификации программного обеспечения

Синтетические методы верификации программного обеспечения основываются на методах статического анализа, тестирования и формальных методах.

Синтетические методы включают виды:

- тестирование на основе моделей (model based testing);
- мониторинг формальных свойств (runtime verification, passive testing);
- статический анализ формальных свойств (static checking);
- синтетические методы структурного тестирования.

Тестирование на основе моделей использует для построения тестов формальные модели требований к программному обеспечению и принятых проектных решений.

Мониторинг формальных свойств осуществляет встраивание проверок формальных свойств программного обеспечения в систему мониторинга. При выполнении мониторинга программного обеспечения контролируются выделенные свойства.

Статический анализ формальных свойств включает расширенный статический анализ и методы на основе проверки свойств моделей.

Синтетические методы структурного тестирования используют статический анализ кода, формальный анализ, мониторинг выполнения построенных тестов [10].

Контрольные вопросы

1 Что называется верификацией программного обеспечения?

2 Что называется аттестацией программного обеспечения?

3 В чём заключается отличие верификации программного обеспечения от аттестации?

4 Что называется качеством программного обеспечения?

5 Какие факторы влияют на качество программного обеспечения?

6 Что называется жизненным циклом программного обеспечения?

7 Какие существуют виды процессов жизненного цикла программных средств?

8 Какие применяются модели жизненного цикла программных средств?

9 Какие используются методы верификации программного обеспечения?

10 Какие выполняются виды экспертизы программного обеспечения? В чём преимущества и недостатки?

11 Какие этапы включает алгоритм метода оценки программного обеспечения по Фагану?

12 Какие существуют виды формальных методов? В чём преимущества и недостатки?

13 Какие используются виды методов статического анализа?

14 Какие применяются виды динамических методов верификации программного обеспечения?

15 Какие существуют виды синтетических методов верификации программного обеспечения?

2 Обеспечение качества разработки программного приложения

2.1 Модель качества программного обеспечения

Модель качества программного обеспечения включает структурные составляющие:

- характеристики качества программного обеспечения;
- атрибуты;
- метрики;
- оценки значений атрибутов.

Характеристики качества программного обеспечения:

- функциональность – набор свойств, обусловливающий способность выполнять функции в соответствии с назначением;
- надёжность – набор свойств, обусловливающий способность сохранять работоспособность за определённый период времени;
- удобство – набор свойств, обусловливающий способность простого освоения программы, подготовки данных и внесение изменений в программу;
- эффективность – набор свойств, обусловливающий способность программного продукта обеспечивать требуемый уровень производительности в соответствии с выделенными ресурсами, временем и другими условиями;
- сопровождаемость – набор свойств, обусловливающий способность адаптации к диагностике отказов и последствий внесения изменений;
- переносимость – набор свойств, обусловливающий способность адаптации к новой среде функционирования.

Атрибуты характеристик качества детализируют показатели качества. Набор атрибутов характеристик качества используется при оценке качества.

Метрики определяются как комбинации методов измерения атрибутов и шкалы измерения значений атрибутов. Для оценки атрибутов качества программного обеспечения применяются метрики с заданным оценочным весом. Согласно стандарту ISO/IEC 9126 Information Technology – Software quality Characteristics and metrics (Part 1-4) метрики определяются по модели измерения атрибутов программного обеспечения на всех этапах жизненного цикла.

Существует три типа метрик:

- метрики программного продукта, которые используются при измерении его характеристик;
- метрики процесса, которые используются при измерении свойства процесса жизненного цикла создания продукта.

- метрики использования, которые оценивают результаты эксплуатации программы.

Оценки значений атрибутов – оценочные элементы метрик, которые используются для оценки количественного или качественного значения отдельного атрибута показателя качества программного обеспечения. В зависимости от назначения, особенностей и условий сопровождения программного обеспечения выбираются наиболее важные характеристики качества и их атрибуты.

Модель качества программного обеспечения изображена на рисунке 2.1.

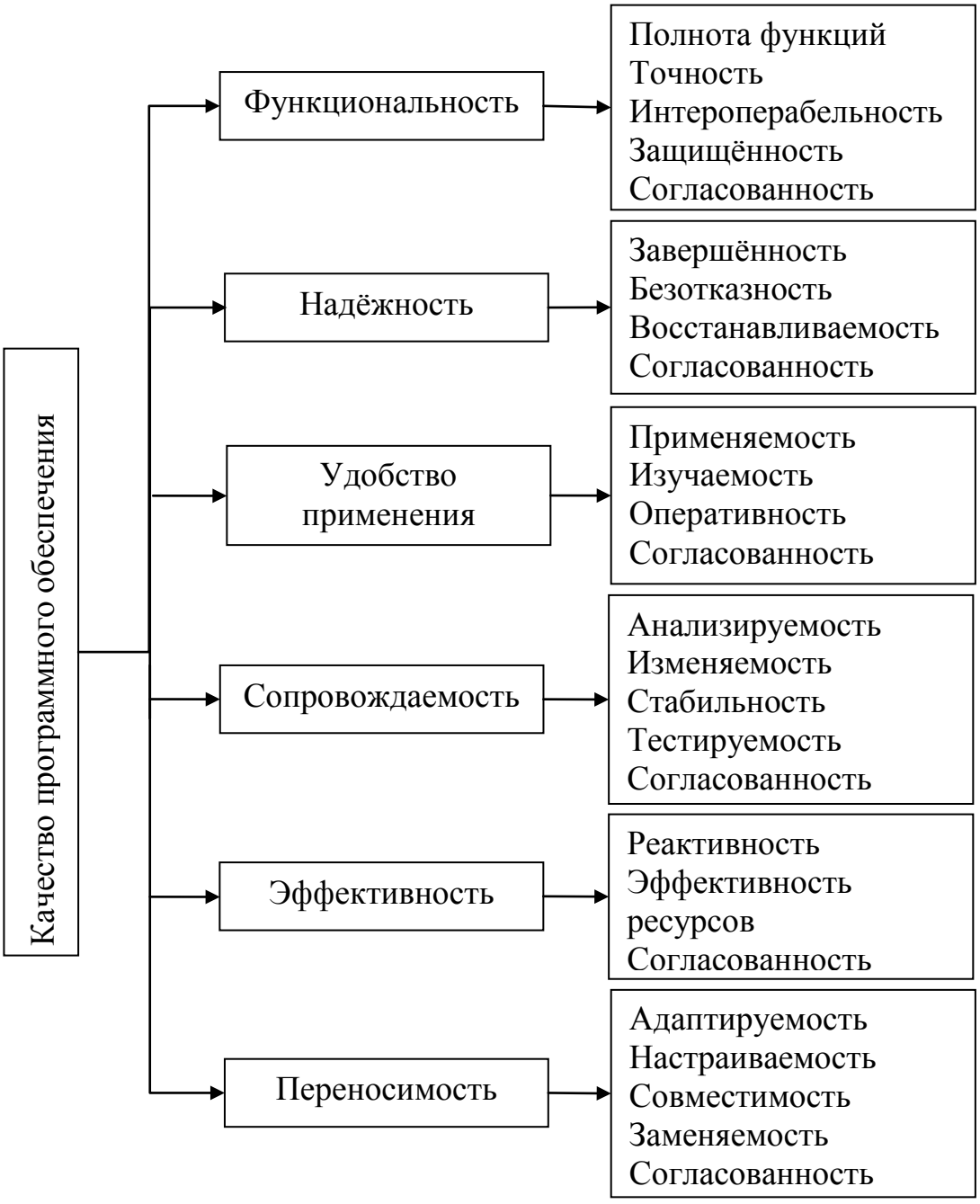


Рисунок 2.1 – Модель качества программного обеспечения

Функциональность включает атрибуты:

- функциональная полнота – свойство компонента, которое показывает степень достаточности основных функций для решения задач в соответствии с назначением программного обеспечения;
- точность – атрибут, который показывает степень достижения правильных результатов;
- интероперабельность – атрибут, который показывает возможность воздействовать на программное обеспечение специальными системами и средами;
- защищённость – атрибут, который показывает на способность программного обеспечения предотвращать несанкционированный доступ к программам и данным.

Атрибуты надёжности подразделяются на виды:

- безотказность – атрибут, который определяет способность программного обеспечения функционировать без отказов;
- устойчивость к ошибкам – атрибут, который показывает на способность программного обеспечения выполнять функции при аномальных условиях (сбой аппаратуры, ошибки в данных и интерфейсах, нарушение в действиях оператора);
- восстанавливаемость – атрибут, который показывает на способность программы к перезапуску для повторного выполнения и восстановления данных после отказов.

Удобство применения включает атрибуты:

- принимаемость – атрибут, который определяет усилия, затрачиваемые на распознавание логических концепций и условий применения программного обеспечения;
- изучаемость (легкость изучения) – атрибут, который определяет усилия пользователей на определение применимости программного обеспечения путем использования операционного контроля, диагностики, а также процедур, правил и документации;
- оперативность – атрибут, который показывает на реакцию системы при выполнении операций и операционного контроля;
- согласованность – атрибут, который показывает соответствие разработки требованиям стандартов, соглашений, правил, законов и предписаний.

Атрибуты сопровождаемости подразделяются на виды:

- анализируемость – атрибут, определяющий необходимые усилия для диагностики отказов или идентификации частей, которые будут модифицироваться;
- изменяемость – атрибут, который определяет удаление ошибок в программном обеспечении или внесение изменений для их устранения, а

также введение новых возможностей в программное обеспечение или в среду функционирования;

- стабильность – атрибут, указывающий на постоянство структуры и риск её модификации;
- тестируемость – атрибут, показывающий на усилия при проведении валидации и верификации с целью обнаружения несоответствий требованиям, а также на необходимость проведения модификации программного обеспечения и сертификации;
- согласованность – атрибут, который показывает соответствие данного атрибута соглашениям, правилам и предписаниям стандарта.

Атрибуты эффективности подразделяются на виды:

- реактивность – атрибут, который показывает время отклика, обработки и выполнения функций;
- эффективность ресурсов – атрибут, показывающий количество и продолжительность используемых ресурсов при выполнении функций программного обеспечения;
- согласованность – атрибут, который показывает соответствие данного атрибута с заданными стандартами, правилами и предписаниями.

Переносимость включает атрибуты:

- адаптивность – атрибут, определяющий усилия, затрачиваемые на адаптацию к различным средам;
- настраиваемость – атрибут, который определяет необходимые усилия для запуска данного программного обеспечения в специальной среде;
- совместимость – атрибут, который определяет возможность использования специального программного обеспечения в среде действующей системы;
- заменяемость – атрибут, который обеспечивают возможность интероперабельности при совместной работе с другими программами с необходимой инсталляцией или адаптацией программного обеспечения;
- согласованность – атрибут, который показывает на соответствие стандартам или соглашениям по обеспечению переноса программного обеспечения [11].

2.2 Метрики лексического анализа программ

Лексический анализ текста программы – процесс определения длины реализации и объема программы на основе анализа текста программы посредством подсчёта количества операндов, операторов и числа их вхождений в текст программы.

Для оценки характеристик программ на основе лексического анализа применяются метрики Холстеда, Джилба и Чепина.

2.2.1 Метрики Холстеда

При проведении лексического анализа текста программы определяются длина реализации и объём программы.

Основные метрические характеристики программы:

- η_1 – число простых (уникальных) операторов в программе;
- η_2 – число простых (уникальных) операндов в программе;
- N_1 – общее число всех операторов в программе;
- N_2 – общее число всех операндов в программе;
- f_{1j} – число появлений в программе j-го оператора, $j = \overline{1, \eta_1}$;
- f_{2j} – число появлений в программе j-го операнда, $j = \overline{1, \eta_2}$;

На основе метрических характеристик программы определяются словарь, длина реализации программы и длина программы.

Количество слов в словаре определяется по формуле

$$\eta = \eta_1 + \eta_2, \tag{2.1}$$

где η_1 – число простых (уникальных) операторов в программе;

η_2 – число простых (уникальных) операндов в программе.

Длина реализации программы определяется по формуле

$$N = N_1 + N_2, \tag{2.2}$$

где N_1 – общее число всех операторов в программе;

N_2 – общее число всех операндов в программе.

Длина программы определяется по формуле

$$\tilde{N} = (\eta_1 * \log_2 \eta_1) + (\eta_2 * \log_2 \eta_2), \tag{2.3}$$

где η_1 – число простых (уникальных) операторов в программе;

η_2 – число простых (уникальных) операндов в программе;

Метрики длины программы и длины реализации программы используются для выявления несовершенств программирования:

- наличие последовательности дополняющих друг друга операторов к одному и тому же операнду;
- наличие неоднозначных операндов;

- наличие синонимичных операндов;
- наличие общих подвыражений;
- ненужное присваивание;
- наличие выражений, которые не представлены в свёрнутом виде как произведение множителей.

Объем программы определяется по формуле

$$V = N * \log_2 \eta = \eta * \log_2^2 \eta, \quad (2.4)$$

где N – длина реализации программы;

η – количество слов в словаре.

Соотношение Холстеда имеет вид:

$$N = \eta_1 * \log_2 \eta + \eta_2 * \log_2 \eta \approx \eta_1 * \log_2 \eta_1 + \eta_2 * \log_2 \eta_2 = N_1 + N_2, \quad (2.5)$$

где η_1 – число простых (уникальных) операторов в программе;

η_2 – число простых (уникальных) операндов в программе;

η – количество слов в словаре;

N_1 – общее число всех операторов в программе;

N_2 – общее число всех операндов в программе.

Взаимная связь между длиной реализации программы и размером словаря определяется по формуле

$$N \approx \eta * \log_2 \eta, \quad (2.6)$$

где η – количество слов в словаре.

Взаимная связь между величиной словаря и объёмом программы определяется по формуле

$$V = \eta * \log_2^2 \eta, \quad (2.7)$$

где η – количество слов в словаре [3].

2.2.2 Метрики Джилба

В качестве меры логической сложности Джилб предложил число логических «двоичных принятий решений». Логическая сложность является определяющим фактором для оценки стоимости программы на начальных

этапах проектирования. Логическую сложность программы Джилб определяет как насыщенность программы условными операторами if-then-else и операторами цикла.

Характеристики программного средства:

- CL – абсолютная сложность программы, характеризуемая количеством операторов условий;
- cl – относительная сложность программы, определяющая насыщенность программы операторами условия;
- количество операторов цикла;
- количество операторов условия;
- число модулей или подсистем;
- отношение числа связей;
- отношение числа ненормальных выходов из множества операторов к общему числу операторов.

Программную надёжность Джилб предлагает рассчитывать как единицу минус отношение числа логических сбоев к общему числу запусков.

Точность определяется как отношение количества правильных данных ко всей совокупности данных. Прецизионность определяется как мера того, насколько часто появляются ошибки, вызванные одинаковыми причинами. Прецизионность рассчитывается как отношение числа фактических ошибок на входе к общему числу наблюдённых ошибок, причинами которых явились эти ошибки на входе [3].

2.2.3 Метрики Чепина

Метод Чепина состоит в оценке информационной прочности программного модуля на основе результатов анализа характера использования переменных, входящих в состав списка ввода и вывода.

Множество переменных, составляющих список ввода-вывода, разбивается на четыре группы:

- P – вводимые переменные для расчётов и для обеспечения вывода;
- M – модифицируемые, создаваемые внутри программы, переменные;
- C – переменные, участвующие в управлении работой программного модуля;
- T – неиспользуемые в программе переменные.

Выражение для определения метрики Чепина имеет вид

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T, \qquad (2.8)$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты.

Весовые коэффициенты применяются для отражения влияния на сложность программы функциональной группы переменных.

С учётом весовых коэффициентов расчётное выражение метрики Чепина имеет вид

$$Q = 1 * P + 2 * M + 3 * C + 0{,}5 * T, \qquad (2.9)$$

где P – вводимые переменные;
M – модифицируемые переменные;
C – переменные управления;
T – неиспользуемые переменные.

2.3 Метрики структурной сложности программ

Структурная сложность программ определяется:

- количеством взаимодействующих компонентов;
- числом связей между компонентами;
- сложностью взаимодействия между компонентами.

Поведение выполняющейся программы и связь между входными и выходными данными определяется набором маршрутов управляющего потокового графа. Управляющий потоковый граф – множество вершин и дуг графа управления программой. Сложность программных модулей связана с количеством и сложностью маршрутов выполнения.

Метрика структурной сложности программ используется для оценки трудоёмкости тестирования и сопровождения модуля, оценки надёжности функционирования.

Маршруты выполнения программы подразделяются на два вида:

- вычислительные маршруты;
- маршруты принятия логических решений.

Вычислительные маршруты – маршруты арифметической обработки данных, предназначенные для преобразования величин, являющихся результатами измерения характеристик.

Сложность вычислительных маршрутов определяется по формуле

$$S_1 = \sum_{i=1}^{m} \sum_{j=1}^{l_i} \nu_j, \qquad (2.10)$$

где m – количество маршрутов выполнения программы;

l_i – число данных, обрабатываемых в i-м маршруте;

ν_j – число значений обрабатываемых данных j-го типа, $2 \le \nu_j \le 5$.

Маршруты принятия логических решений – это маршруты управляющего потокового графа программы, отражающие логику выполнения программы, которая может изменять последовательность выполнения команд, переводить управление на удалённые участки программы или завершать выполнение.

Сложность маршрутов принятия логических решений определяется по формуле

$$S_2 = \sum_{i=1}^{m} p_i \,, \tag{2.11}$$

где p_i – число ветвлений или число проверяемых условий в i-м маршруте.

Общая сложность программы определяется по формуле

$$S = c * \sum_{i=1}^{m} \left(\sum_{j=1}^{l_i} v_i + p_i \right) \,, \tag{2.12}$$

где c – коэффициент пропорциональности, корректирующий взаимное использование метрик сложности вычислительных маршрутов и маршрутов принятия логических решений.

2.3.1 Критерии формирования маршрутов

Поток управления – последовательность выполнения модулей и операторов программы.

Граф потока управления (управляющий граф) – ориентированный граф, моделирующий поток управления программой.

Оценка структурной сложности программы определяется по критериям:

- критерий 1;
- критерий 2;
- критерий 3.

Структурная сложность по первому критерию вычисляется по формуле 2.11. Критерий 1 предполагает, что граф потока управления программой должен быть проверен по минимальному набору маршрутов, проходящих через каждый оператор ветвления по каждой дуге. Прохождение по маршруту выполняется один раз, повторная проверка дуг не проводится и считается избыточной. В процессе проверки гарантируется вы-

полнение всех передач управления между операторами программы и каждого оператора не менее одного раза [3].

Критерий 2 основан на анализе базовых маршрутов в программе, которые формируются и оцениваются на основе цикломатического числа графа потока управления программой.

Цикломатическое число графа потока управления программой определяется по формуле

$$Z = m - n + 2 * p, \qquad (2.13)$$

где m – общее число дуг в графе;

p – число связных компонентов графа.

Число связных компонентов графа p равно количеству дуг, необходимых для превращения исходного графа в максимально связный граф – граф потока управления, у которого любая вершина доступна из любой другой вершины.

Второй критерий, выбранный по цикломатическому числу, требует однократной проверки или тестирования каждого линейно независимого цикла и каждого линейно независимого ациклического участка. При использовании второго критерия количество проверяемых маршрутов равно цикломатическому числу.

Для автоматического анализа графов по второму критерию используются матрицы смежности и достижимости графов, содержащие информацию о структуре проверяемой программы.

Третий критерий формирования маршрутов для тестирования программ основан на формировании полного состава базовых структур графа потока управления программой и заключается в анализе каждого из реальных ациклических маршрутов исходного графа программы и каждого цикла. Каждый компонент структуры программы должен быть проанализирован хотя бы один раз. Если при прохождении ациклического маршрута исходного графа потока управления достижимы несколько элементарных циклов, то при тестировании должны исполняться все достижимые циклы [3; 12].

2.3.2 Метрика Маккейба

Метрика Маккейба характеризует цикломатическое число графа потока управления программой и определяется по формуле

$$M = m - n + 2, \qquad (2.14)$$

где m – количество рёбер графа;

n – число вершин графа.

Цикломатическое число Маккейба – величина M, рассчитанная по формуле 2.14.

Цикломатическая сложность программы – структурная мера сложности программы, применяемая для измерения качества программного обеспечения и основанная на методах статического анализа кода программных средств.

При определении цикломатической сложности используется граф потока управления программой: узлы графа соответствуют неделимым группам команд программы и ориентированным рёбрам, каждый из которых соединяет два узла и соответствует двум командам. Цикломатическая сложность может быть применена для отдельных функций, модулей, методов или классов в пределах анализируемого программного средства. Эта стратегия тестирования называется основным маршрутом тестирования Маккейба. Тестирование представляет собой проверку линейного независимого маршрута графа программы. Количество тестов должно быть равно цикломатической сложности.

Теоретической основой определения цикломатического числа Маккейба является теория графов [3; 12].

2.3.3 Методика построения графов управления

Ориентированный управляющий граф учитывает логику программы и обеспечивает проведение анализа потока передачи управления между операторами. Граф формируется из вершин, которые соответствуют операторам программы, и дуг, задающих переходы между операторами.

Этапы построения графов управления:

1 Учитываются исполнимые операторы и не учитываются операторы описания.

2 Операторы, не изменяющие порядок действий в программе и следующие друг за другом, объединяются в одну вершину графа управления.

3 Операторы цикла заменяются несколькими вершинами.

4 В графе должны быть три группы вершин, определяющих три элемента, составляющих собой цикл: начало цикла, тело цикла, ветвление при окончании цикла – завершение или возвращение к исходному оператору.

Правильно построенный управляющий граф позволяет оценить структурную сложность программы, определяемую количеством независимых контуров в полносвязном графе [12].

2.4 Процедурно-ориентированные метрики

2.4.1 Оценка качества программы на основе функциональных указателей и метрик свойств

Оценка качества процедурно-ориентированных программных средств производится на основе функциональных указателей (FP – functions points) и указателей свойств (FP – features points).

Функциональные указатели характеризуют функциональную сложность программы. Количество функциональных указателей определяется количеством ссылок на внешний ввод, внешний вывод, внешний запрос, автономный локальный файл, глобальный файл.

Количество функциональных указателей определяется по формуле

$$FP = F * (0{,}65 + 0{,}01 * \sum_{i=1}^{14} k_i), \qquad (2.15)$$

где k_i – коэффициент регулировки сложности, зависящий от ответов на вопросы, связанные с особенностями программы, $i = \overline{1{,}14}$.

F – общее количество функциональных указателей.

Общее количество функциональных указателей определяется по формуле

$$F = \sum_{j=1}^{5} f_j, \qquad (2.16)$$

где f_j – оценочные элементы, $j = \overline{1{,}5}$.

Оценочные элементы подразделяются на виды:

- f_1 – количество внешних вводов (вводов данных пользователем);
- f_2 – количество внешних выводов (отчёты, экраны, распечатки, сообщения);
- f_3 – количество внешних запросов (диалоговых вводов-выводов);
- f_4 – количество локальных внутренних логических файлов (группа логически связанных данных, которая размещена внутри приложения и обслуживается через внешние вводы, базы данных);
- f_5 – количество внешних интерфейсных файлов, разделяемых с другими программами глобальных файлов (группа логически связанных

данных, которая размещается и поддерживается внутри другого приложения, внутренний логический файл в другом приложении).

Значения коэффициентов регулировки сложности k_i зависят от ответов на вопросы, касающиеся влияния определенных факторов на выполнение функций программного обеспечения:

1 Какое влияние имеет наличие средств передачи данных?

2 Какое влияние имеет распределенная обработка данных?

3 Какое влияние имеет распространенность используемой аппаратной платформы?

4 Какое влияние имеет критичность к требованиям производительности и ограничению времени ответа?

5 Какое влияние имеет частота транзакций?

6 Какое влияние имеет ввод данных в режиме реального времени?

7 Какое влияние имеет эффективность работы конечного пользователя?

8 Какое влияние имеет оперативное обновление локальных файлов в режиме реального времени?

9 Какое влияние имеет скорость обработки данных?

10 Какое влияние имеют количество и категории пользователей?

11 Какое влияние имеет легкость инсталляции?

12 Какое влияние имеет легкость эксплуатации?

13 Какое влияние имеет разнообразие условий применения?

14 Какое влияние имеет простота внесения изменений?

Ответы на каждый вопрос должны быть даны в соответствии со следующими категориями ответов, касающихся влияния конкретных факторов:

- влияние случайное;
- влияние небольшое, эпизодическое;
- влияние среднее;
- влияние важное, значительное;
- влияние основное, существенное.

Значения весовым коэффициентам важности назначают следующим образом:

- $k_i = 1$, если на i-й вопрос выбран ответ «влияние случайное»;
- $k_i = 2$, если на i-й вопрос выбран ответ «влияние небольшое»;
- $k_i = 3$, если на i-й вопрос выбран ответ «влияние среднее»;
- $k_i = 4$, если на i-й вопрос выбран ответ «влияние важное»;
- $k_i = 5$, если на i-й вопрос выбран ответ «влияние основное».

Оценка качества программных средств определяется по формуле

$$DQ = \frac{Количеств\ оошибок}{\sum_{j=1}^{5} f_j * (0.65 + 0.01 * \sum_{i=1}^{14} k_i)}, \qquad (2.17)$$

где f_j – оценочные элементы, $j = \overline{1,5}$;

k_i – коэффициент регулировки сложности, зависящий от ответов на вопросы, связанные с особенностями программы, $i = \overline{1,14}$

После вычисления количества функциональных указателей FP на его основе формируются косвенные метрики:

- производительность;
- качество;
- удельная стоимость;
- документированность.

Производительность определяется как отношение количества функциональных указателей к затратам. Единицы измерения производительности – FP/чел.-мес.

Качество рассчитывается как отношение количества ошибок к количеству функциональных указателей. Единицы измерения качества – единицы/FP.

Удельная стоимость определяется как отношение стоимости к количеству функциональных указателей. Единицы измерения удельной стоимости – тыс. руб./FP.

Документированность рассчитывается как отношение количества страниц документа к количеству функциональных указателей. Единицы измерения документированности – количество страниц/FP.

Функциональные указатели применяются в коммерческих информационных системах.

Оценка качества программных средств с высокой алгоритмической сложностью производится посредством метрики свойств (features points). Они применяются в системном и инженерном программном обеспечении, встроенном программном обеспечении. Для вычисления указателя свойств используется характеристика – количество алгоритмов. Для формирования указателя свойств составляется таблица. Расчёт указателя свойств производится по формуле расчёта функциональных указателей (2.15) [3; 15].

2.4.2 Оценка качества программы на основе связности программных модулей

Программный модуль – часть программы, оформленная как самостоятельный программный продукт, который может быть применён для использования в описаниях проектируемого процесса.

Программные модули позволяют снизить сложность программы и исключить повторение ранее проделанных действий при программировании.

При использовании модульной структуры программы необходимо обеспечить связывание программных модулей.

Для оценки качества программного средства используют метрики связности модулей: уровень качества программного продукта прямо пропорционален силе связности (прочности) программных модулей. Для обеспечения надёжности программы разработка качественного программного средства должна максимизировать связность программных модулей.

Связность – мера прочности соединения функциональных и информационных объектов в пределах одного программного модуля.

Типы связности программных модулей:

1 Связность по совпадению (сила связности 0) – отсутствие внутренних связей в программных модулях.

2 Логическая связность (сила связности 1) – связность в программных модулях, содержащих подпрограммы для различных вариантов обращения к модулям.

3 Временная связность (сила связности 3) – связность программных модулей, составные части которых не связаны между собой, но должны быть выполнены в течение определённого периода времени.

4 Процедурная связность (сила связности 5) – связность программных модулей, в которых имеет место порядковая связь составляющих их частей, реализующих процедуру обработки данных.

5 Коммуникационная связность (сила связности 7) – связность программных модулей, составляющие которых связаны по данным (используют одну и ту же структуру данных).

6 Информационная связность (сила связности 9) – связность программных модулей, когда выходные данные одной части модуля используются как входные данные другой части модуля.

7 Функциональная связность (сила связности 10) – связность, при которой модуль как единое целое реализует единственную функцию, так как программный модуль содержит набор объектов, предназначенных для выполнения одной и только одной задачи.

Связность типов 1, 2, 3, 4 свидетельствует о недостатках проектирования архитектуры программы с точки зрения тестирования и последующего сопровождения.

Программные модули с высокой функциональной связностью создают хорошие предпосылки для применения прогрессивной технологии повторного использования в процессе будущей разработки программных комплексов.

Для определения типа связности программных модулей применяется методика:

1 Если модуль реализует единственную прикладную функцию, то тип связности – функциональный. Далее анализ можно прекратить (конец процедуры). В противном случае следует перейти к пункту 2.

2 Если действия внутри модуля связаны, то нужно перейти к пункту 3. Если же действия внутри модуля ничем не связаны, то следует перейти к пункту 5 для дополнительного анализа.

3 Если действия внутри модуля связаны по данным, то перейти к пункту 4, а если действия внутри модуля связаны потоком управления – перейти к пункту 6.

4 Если порядок действий внутри модуля важен, то тип связности информационный, в противном случае тип связности – коммуникационный.

Анализ прекращается, и осуществляется переход к концу процедуры.

5 Если действия внутри модуля принадлежат к одной категории, то уровень связности – логический. Если действия внутри модуля не принадлежат к одной категории, то модуль обладает типом связности по совпадению. Далее следует переход к концу процедуры.

6 Если порядок действия внутри программного модуля важен, то тип связности – процедурный. В противном случае тип связности – временной. Анализ прекращается [3; 12].

2.4.3 Оценка качества программы на основе сцепления программных модулей

Сцепление – мера межмодульной связи, которую для повышения качества программных средств необходимо уменьшать.

Для оценки качества программного средства используют метрики сцепления модулей: уровень качества программного продукта обратно пропорционален силе сцепления программных модулей. Для обеспечения надёжности программы разработка качественного программного средства должна минимизировать сцепление программных модулей.

Для измерения сцепления используют целочисленную шкалу степени сцепления в интервале значений от 0 до 9. Для каждой степени сцепления определен тип сцепления, который может быть сопоставлен модулю в процессе проектирования.

Типы сцепления программных модулей:

1 Сцепление по данным (сила сцепления 1) – сцепление, при котором модуль является вызываемым, входные параметры представляются простыми элементами данных.

2 Сцепление по образцу (сила сцепления 3) – сцепление, при котором модуль является вызываемым, а в качестве параметров используются агрегатные типы данных.

3 Сцепление по управлению (сила сцепления 4) – сцепление, при котором модуль является вызывающим и передаёт вызываемому модулю список управляющих параметров, явно влияющих на его работу.

4 Сцепление по внешним ссылкам (сила сцепления 5) – сцепление, при котором модуль имеет адресные ссылки (указатели) на глобальный элемент данных, на который ссылается другой программный модуль.

5 Сцепление по общей области (сила сцепления 7) – сцепление, при котором модуль разделяет (совместно использует) с другим модулем одну и ту же глобальную структуру данных.

6 Сцепление по содержанию (сила сцепления 9) – сцепление, при котором модуль напрямую ссылается на содержимое другого модуля.

Хорошо спроектированная система характеризуется слабым сцеплением программных модулей.

Методы достижения слабого сцепления программных модулей:

- удаление необязательных связей;
- снижение количества необходимых связей;
- упрощение необходимых связей.

Способы снижения сцепления программных модулей:

1 Создание минимальных по количеству параметров межмодульных связей.

2 Создание прямых межмодульных связей.

3 Создание локализованных связей.

4 Создание явных связей.

5 Создание гибких связей для облегчения модификаций [3; 12].

2.5 Объектно-ориентированные метрики

Для объектно-ориентированных метрик целесообразно представление абстракций в терминологии измерения класса.

2.5.1 Метрики Мартина

Для оценки характеристик объектно-ориентированных программ используются метрики Мартина:

1 Центростремительное сцепление Ca – метрика, определяющая количество классов вне конкретной категории, которые зависят от классов внутри нее.

2 Центробежное сцепление Ce – метрика, оценивающая количество классов внутри конкретной категории, которые зависят от класса вне неё.

3 Нестабильность I – расчётная метрика, значение которой определяется по формуле

$$I = \frac{C_e}{C_a + C_e}, \tag{2.18}$$

где C_a – центростремительное сцепление;

C_e – центробежное сцепление.

4 Абстрактность A – мера, оценивающая абстрактность категории. Если категория абстрактна, то она является достаточно гибкой и может быть расширена.

Абстрактность определяется по формуле

$$A = \frac{n_A}{n_{All}}, \tag{2.19}$$

где n_A – количество абстрактных классов в категории;

n_{All} – общее количество классов в категории.

Значения метрики абстрактности располагаются в диапазоне от 0 до 1. Категория конкретна при нулевом значении абстрактности. Категория абстрактна при единичном значении.

На основе метрик Мартина строится график, отражающий зависимость между абстрактностью и нестабильностью (главная последовательность).

5 Расстояние до главной последовательности определяется по формуле

$$D = \left| \frac{A + I - 1}{\sqrt{2}} \right|, \tag{2.20}$$

где A – абстрактность;

I – нестабильность.

6 Нормализованное расстояние до главной последовательности определяется по формуле

$$D_n = |A + I - 2|, \qquad (2.21)$$

где A – абстрактность;

I – нестабильность.

Чем ближе классы находятся к главной последовательности, тем лучше для обеспечения качества программного средства [3; 12].

2.5.2 Метрики Чидамбера и Кемерера

Метрики Чидамбера и Кемерера основаны на анализе методов класса, дерева наследования и включают шесть метрик:

1 *Взвешенные методы на класс WMC* (Weighted Methods Per Class). Данная метрика позволяет измерять сложность классов с учетом сложности их методов. Метрика называется взвешенной, потому что весом метода считается количественная характеристика сложности метода.

Пусть в классе C определены n методов со сложностью c[1], c[2], …, c[n]. Для оценки сложности может быть выбрана любая метрика сложности (например, Холстеда или цикломатическая сложность в зависимости от интересующего критерия). Основным при этом является процесс нормализации этой метрики таким образом, чтобы номинальная сложность для метода принимала значение 1. В этом случае количество методов и их сложность могут служить своеобразным индикатором затрат, необходимых на реализацию и тестирование классов.

2 *Глубина дерева наследования DIT* (Depth of Inheritance Tree) позволяет определить количество классов-предков, которые потенциально оказывают влияние на данный класс. Эта метрика характеризует самый длинный путь по иерархии классов к данному классу от класса-предка. Этот показатель должен быть возможно большим, так как при большей глубине возрастает абстракция данных, снижается насыщенность класса методами.

3 *Количество потомков NOC* (Number Of Child) позволяет определить количество непосредственных потомков данного класса. С ростом значения *NOC* возрастает многократность использования, поскольку наследование представляет собой одну из форм повторного использования. В то же время при возрастании метрики *NOC* снижается уровень абстракции базового класса.

4 *Связанность между классами объектов CBO* (Coupling Between Object classes) дает возможность определить количество классов, с которыми связан данный класс. Это имеет существенное значение, когда один класс использует методы или экземпляры другого класса. Данная метрика характеризует статическую составляющую внешних связей классов. С ростом значения *CBO* уменьшается абстракция данных, и многократное использование класса уменьшается.

5 *Количество откликов на класс RFC* (Response For Class) позволяет определить количество методов, которое может быть выполнено в ответ на получение сообщения данным классом. Метрика *RFC* является мерой потенциального взаимодействия данного классах другими классами и позволяет судить о динамике поведения объекта в системе, т. е. данная метрика характеризует динамическую составляющую внешних связей классов.

6 *Отсутствие сцепления в методах LCOM* (Lack Cohesion Of Methods) позволяет оценить зависимость методов класса друг от друга. Для вычисления этой метрики подсчитывается количество пар методов, которые не используют общие атрибуты класса. Затем подсчитывается количество пар методов, которые используют общие переменные. Метрика *LCOM* равна разности между первым числом и вторым. Если при этом получается отрицательное число, то значение метрики считается равным нулю [3; 12].

2.5.3 Метрики Лоренца и Кидда

Метрики Лоренца и Кидда подразделяются на группы:

- метрики размера, основанные на подсчёте свойств и операций классов, средних значений объектно-ориентированной программы;
- метрики наследования, учитывающие способы повторного использования операций в иерархии классов;
- внутренние метрики, отвечающие на вопросы связности и кодирования;
- внешние метрики, изучающие сцепление и повторное использование

Метрики Лоренца и Кидда включают десять метрик:

1 *Размер класса* CS (Class Size) – общий размер класса определяется на основании определения следующих показателей:

- общее количество операций;
- количество свойств.

Указанные измерения в обоих случаях следует проводить с учетом приватных и наследуемых экземплярных операций, которые инкапсулируются внутри класса:

$$CS = C_{\sum} + S_{\sum}, \qquad (2.22)$$

где $C_{\sum}$ – количество инкапсулированных классом методов (операций);

$S_{\sum}$ – количество инкапсулированных классом свойств.

2 *Количество операций, переопределяемых подклассом NOO* (Number of Operations Overridden by a Subclass). Переопределением назы-

вают случай, когда подкласс замещает операцию, унаследованную от суперкласса, своей собственной версией. Большие значения *N00* указывают на возникшие проблемы проектирования.

3 *Количество операций, добавленных подклассом NOA* (Number of Operations Added by a Subclass), определяется количеством добавленных относительно родительского класса собственных методов (операций):

$$NOA = N_{\sum}, \tag{2.23}$$

где $N_{\sum}$ – количество новых методов класса, добавленных относительно суперкласса.

С увеличением *NOA* подкласс приобретает меньшую общность со своим суперклассом, что требует больших трудозатрат по тестированию и внесению изменений. При увеличении высоты дерева иерархии классов (с ростом значения *DIT)* должно уменьшаться количество новых методов классов нижних уровней. Для рекомендуемых граничных значений размера класса (*CS* = 20) и высоты дерева иерархии классов (*DIT*= 6) значение *NOA* ограничено значением 4 *(NOA* < 4).

4 *Индекс специализации SI* (Specialization Index) характеризует грубую оценку степени специализации каждого подкласса при добавлении, удалении или переопределении операций. Индекс специализации определяется по формуле

$$SI = \frac{NOO * u}{M_{общ}}, \tag{2.24}$$

где NOO – количество операций, переопределяемых подклассом;

u – номер уровня в иерархии, на котором находится подкласс;

$M_{общ}$ – общее количество методов класса.

Чем выше значение метрики *SI*, тем выше вероятность того, что в иерархии классов есть отдельные экземпляры, нарушающие абстракцию суперкласса. Рекомендуемое значение показателя *SI* ограничено сверху величиной 0,15, т. е. *SI* < 0,15.

5 *Средний размер операции AOS* (Average Operation Size) определяется количеством сообщений, порождаемых операцией. В качестве оценки размера может использоваться количество строк программы, однако LOC-оценки приводят к известным проблемам. Иным вариантом может являться количество сообщений, посланных операцией. Рост значения данного показателя означает, что обязанности размещены в классе не очень удачно. Рекомендуемое значение метрики *AOS* не должно превышать 9. Уве-

личение среднего размера относительно этой границы рассматривают как показатель неудачного проектирования обязанностей класса.

6 *Сложность операции OC* (Operation Complexity) может быть вычислена на основе стандартных метрик сложности (например, с помощью *LOC*- или *FP*-оценок, метрики цикломатической сложности, метрики Холстеда). Лоренц и Кидд предложили вычислять значение OC суммированием оценок с весовыми коэффициентами, приведенными в таблице 2.1.

Таблица 2.1 – Весовые коэффициенты

Действие	Вес
Определение переменной-параметра	0,3
Определение временной переменной	0,5
Присваивание значения	0,5
Вложенное выражение	0,5
Сообщение без параметра	1
Арифметическая операция	2
Сообщение с параметрами	3
Вызов стандартной функции интерфейса	5
Вызов пользовательской функции	7

Рекомендуемое значение метрики ограничено числом 65 *(OC* < 65).

7 *Среднее количество параметров на операцию ANP* (Average Number of Parameters per operation) определяется отношением числа параметров к количеству операций (методов) класса. Чем больше параметров у операции, тем сложнее взаимодействие между объектами. Поэтому значение метрики *ANP* должно быть как можно меньшим. Рекомендуемое значение *ANP* = 0,7.

8 *Количество описаний сценариев NSS* (Number of Scenario Scripts) измеряется или количеством классов, реализующих требования к программному обеспечению, или количеством состояний для каждого класса, или количеством методов класса. При своем не совсем обычном способе измерения метрика *NSS* является достаточно эффективным индикатором размера создаваемой программы. Рекомендуется не менее одного сценария. Рост количества сценариев неминуемо ведет к увеличению размера программы.

9 *Количество ключевых классов NKC* (Number of Key Classes) характеризует объем работы по программированию. Рекомендуется ограничивать значения метрики снизу значением 0,2. Если значение метрики NKC<0,2 от общего количества классов системы, то необходимо пересмотреть выделение классов.

10 *Количество подсистем NSUB* (Number of subsystem) определяется непосредственным подсчетом. Количество подсистем обеспечивает понимание таких вопросов, как размещение ресурсов, планирование, общие затраты на интеграцию. Рекомендуется выделять в программном комплексе не менее трёх подсистем [3; 12].

2.5.4 Метрики Абреу

Набор метрик Абреу (MOOD, Metrics for Object Oriented Design) включает в себя показатели качества программных средств:

- фактор закрытости метода (MHF);
- фактор закрытости свойства (AHF);
- фактор наследования метода (MIF);
- фактор наследования свойства (AIF)
- фактор полиморфизма (POF);
- фактор сцепления (COF).

Метрики MHF и AHF относятся к первому принципу объектно-ориентированного программирования – инкапсуляции. Метрики MIF и AIF относятся ко второму принципу объектно-ориентированного программирования – наследованию. Метрики POF относятся к третьему принципу объектно-ориентированного программирования – полиморфизму. Метрики COF относятся к отсылке сообщений.

Показатели качества метрик Абреу:

1 Фактор закрытости метода MHF (Method Hiding Factor) – показатель, характеризующий процентное количество классов, из которых метод невидим.

Значение показателя MHF определяется по формуле

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}, \qquad (2.25)$$

где $M_h(C_i)$ – реализация класса, характеризующая количество скрытых методов в классе;

$M_d(C_i)$ – общее количество методов, определённых в классе (не учитываются наследованные методы);

C_i – класс с номером i, $i = \overline{1, TC}$;

TC – количество классов в создаваемой программной системе.

Общее количество методов в классе определяется по формуле

$$M_d(C_i) = M_v(C_i) + M_h(C_i), \quad (2.26)$$

где $M_v(C_i)$ – интерфейс класса, определяющий количество видимых методов в классе.

Видоизменённая формула расчёта показателя MHF имеет вид:

$$MHF = \frac{\sum_{I=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}, \quad (2.27)$$

где $V(M_{mi})$ – процентное количество классов, которые способны видеть m-й метод i-го класса, $i = \overline{1, TC}$;

$M_d(C_i)$ – общее количество методов, определённых в классе (не учитываются наследованные методы);

C_i – класс с номером i, $i = \overline{1, TC}$;

TC – количество классов в создаваемой программной системе.

Процентное количество классов, которые способны видеть m-й метод i-го класса определяется по формуле

$$V(M_{mi}) = \frac{\sum_{i=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1}, \quad (2.28)$$

где $is\_visible(M_{mi,}C_j)$ – уровень видимости m-го метода i-го класса из j-го класса, $i = \overline{1, TC}$.

Уровень видимости m-го метода i-го класса из j-го класса определяется по формуле

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1, \text{если } j \neq 1 \text{ и } C_j \text{ может вызвать } M_{mi} \\ 0 \text{ в остальных случаях} \end{cases}, \quad (2.29)$$

где Ami – m-е свойство i-го класса;

C_j – класс с номером j.

С увеличением значения метрики MHF уменьшается плотность дефектов в системе, что означает сокращение затрат, необходимых для устранения ошибок [3; 12].

2 Фактор закрытости свойства AHF (Attribute Hiding Factor) – показатель, представляющий процентное количество классов, из которых данное свойство невидимо.

Значение показателя AHF определяется по формуле

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}, \qquad (2.30)$$

где $A_h(C_i)$ – реализация класса, характеризующая количество скрытых свойств в классе;

$A_d(C_i)$ – общее количество свойств, определённых в классе (не учитываются наследованные свойства);

C_i – класс с номером i, $i = \overline{1, TC}$;

TC – количество классов в создаваемой программной системе.

Общее количество свойств, определённых в классе C_i определяется по формуле

$$A_d(C_i) = A_v(C_i) + A_h(C_i), \qquad (2.31)$$

где $A_v(C_i)$ – интерфейс класса, определяющий количество видимых свойств в классе. C_i, $i = \overline{1, TC}$;

Видоизменённая формула расчёта показателя AHF имеет вид:

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}, \qquad (2.32)$$

где $V(A_{mi})$ – процентное количество классов, которые видят m-е свойство i-го класса из j-го класса, $i = \overline{1, TC}$;

$A_d(C_i)$ – общее количество свойств, определённых в классе (не учитываются наследованные свойства);

C_i – класс с номером i, $i = \overline{1, TC}$;

TC – количество классов в создаваемой программной системе.

Процентное количество классов, которые видят m-е свойство i-го класса определяется по формуле

$$V(A_{mi}) = \frac{\sum_{i=1}^{TC} is\_visible(A_{mi}, C_j)}{TC - 1}, \quad (2.33)$$

где $is\_visible(A_{mi,}C_j)$ – уровень видимости m-го свойства i-го класса из j-го класса, $i = \overline{1, TC}$.

Уровень видимости m-го свойства i-го класса из j-го класса определяется по формуле

$$is\_visible(A_{mi}, C_j) = \begin{cases} 1, \text{если } j \neq 1 \text{ и } C_j \text{ может вызвать } A_{mi} \\ 0 \text{ в остальных случаях} \end{cases}, \quad (2.34)$$

где Ami – m-е свойство i-го класса;

C_j – класс с номером j.

В числителе формулы 2.32 расположена сумма закрытости всех свойств во всех классах, а в знаменателе – общее количество свойств, определённых в системе. Если все свойства скрыты и доступны для методов класса, то значение фактора закрытости свойств AHF равно 100% [3; 12].

3 Фактор наследования метода MIF (Method Inheritance Factor) – показатель, характеризующий процентное количество классов, из которых метод наследован. Значение фактора наследования метода определяется по формуле

$$MIF = \frac{\sum_{i=1}^{TC} Mi(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}, \quad (2.35)$$

где $M_i(C_i)$ – количество наследованных и непереопределённых методов в классе C_i;

$M_a(C_i)$ – общее количество методов, доступное в классе C_i;

C_i – класс с номером i, $i = \overline{1, TC}$;

TC – количество классов в создаваемой программной системе.

Общее количество методов, доступных в классе C_i, определяется по формуле

$$M_a(C_i) = M_d(C_i) + M_i(C_i), \tag{2.36}$$

где $M_i(C_i)$ – количество наследованных и непереопределённых методов в классе C_i;

$M_d(C_i)$ – количество методов, определённых в классе C_i.

Количество методов, определённых в классе C_i, определяется по формуле

$$M_d(C_i) = M_n(C_i) + M_0(C_i), \tag{2.37}$$

где $M_n(C_i)$ – количество новых (ненаследованных и переопределённых) методов в классе C_i;

$M_0(C_i)$ – количество наследованных и переопределённых методов в классе C_i.

Числитель в формуле 2.35 представляет сумму наследованных и непереопределённых методов во всех классах программного приложения. Знаменатель – общее количество доступных методов (локально определённых и наследованных) для всех классов.

Если значение фактора наследования метода MIF равно нулю, то в анализируемой программной системе отсутствует эффективное наследование. При увеличении значения фактора наследования метода MIF уменьшается плотность дефектов и сокращаются затраты на исправление ошибок [3; 12].

4 Фактор наследования свойства AIF (Atribute Inheritance Factor) показатель, характеризующий процентное количество классов, из которых свойство наследовано.

Значение показателя AIF определяется по формуле

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}, \tag{2.38}$$

где $A_i(C_i)$ – количество наследованных и непереопределённых свойств в классе C_i;

$A_a(C_i)$ – общее количество свойств, доступных в классе C_i;

C_i – класс с номером i, $i = \overline{1, TC}$;

TC – количество классов в программной системе.

Общее количество свойств, доступных в классе C_i, определяется по формуле

$$A_a(C_i) = A_d(C_i) + A_i(C_i), \tag{2.39}$$

где $A_d(C_i)$ – количество свойств, определённых в классе. C_i, $i = \overline{1, TC}$;

$A_i(C_i)$ – количество наследованных и непереопределённых свойств в классе C_i.

Количество свойств, определённых в классе C_i, рассчитывается по формуле

$$A_d(C_i) = A_n(C_i) + A_0(C_i), \tag{2.40}$$

где $A_n(C_i)$ – количество новых (ненаследованных и переопределённых) свойств в классе C_i, $i = \overline{1, TC}$;

$A_0(C_i)$ – количество наследованных и переопределённых свойств в классе C_i.

Числитель в формуле 2.38 представляет сумму наследованных и непереопределённых свойств во всех классах программной системы. Знаменатель – общее количество доступных свойств (локально определённых и наследованных) для всех классов [3; 12].

5 Фактор полиморфизма POF (Polymorphism Factor) – показатель, представляющий отношение реального количества возможных полиморфных ситуаций к максимальному количеству возможных полиморфных ситуаций для класса. Значение фактора полиморфизма POF рассчитывается по формуле

$$POF = \frac{\sum_{i=1}^{TC} M_0(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) * DC(C_i)]}, \tag{2.41}$$

где $M_0(C_i)$ – количество наследованных и переопределённых методов в классе C_i;

$DC(C_i)$ – количество потомков класса C_i;

$M_n(C_i)$ – количество новых методов, определённых в классе C_i.

Количество методов, определённых в классе C_i, определяется по формуле

$$M_d(C_i) = M_n(C_i) + M_0(C_i), \tag{2.42}$$

где $M_n(C_i)$ – количество новых (ненаследованных и переопределённых) методов в классе C_i;

$M_0(C_i)$ – количество наследованных и переопределённых методов в классе C_i, $i = \overline{1,TC}$;

TC – количество классов в программной системе.

Использование полиморфизма уменьшает плотность дефектов и затраты на доработку программных средств. Если значение фактора полиморфизма POF>10%, то количество ошибок в программе и затраты могут увеличиться [3; 12].

6 Фактор сцепления COF (Coupling Factor) – показатель, характеризующий наличие отношения между классами: класс-клиент содержит по меньшей мере одну ненаследованную ссылку на свойство или метод класса-поставщика.

Значение фактора сцепления COF определяется по формуле

$$COF = \frac{\sum_{i=1}^{TC} [\sum_{j=1}^{TC} is\_client(C_i, C_j)]}{TC^2 - TC}, \tag{2.43}$$

где $is\_client(C_i, C_j)$ – бинарная переменная, указывающая на наличие отношения «клиент-поставщик» и принимающая значение 1 или 0;

TC – количество классов в программной системе.

Бинарная переменная, указывающая на наличие отношения «клиент-поставщик», определяется по формуле

$$is\_client(C_C, C_S) = \begin{cases} 1, \text{если } C_C \to C_S \cap C_C \neq C_S \\ 0, \text{в остальных случаях} \end{cases}, \tag{2.44}$$

где $C_C \to C_S$ – отношение «клиент-поставщик», означающее, что класс-клиент содержит по меньшей мере одну ненаследованную ссылку на свойство или метод класса-поставщика.

Числитель формулы расчёта COF содержит реальное количество сцеплений, не относящихся к наследованию. Знаменатель формулы расчёта COF соответствует максимально возможному количеству сцеплений в программной системе с классами.

С увеличением количества сцеплений плотность дефектов и уровень затрат на доработку возрастают. Сцепления отрицательно влияют на качество программного средства и увеличивают сложность программного обеспечения, уменьшают инкапсуляцию и возможность повторного использования программных средств, затрудняет понимание и усложняет сопровождение программ [3; 12].

Контрольные вопросы

1 Что понимается под моделью качества программного обеспечения?

2 Какие существуют структурные составляющие модели качества программного обеспечения?

3 Какие типы метрик используются для оценивания программных продуктов?

4 Что понимается под лексическим анализом текста программ?

5 Какие метрики применяются для оценки характеристик программ на основе лексического анализа?

6 Сколько и какие характеристики метрики Холстеда используются для оценки качества программного обеспечения?

7 Сколько и какие характеристики метрики Джилба используются для оценки качества программного обеспечения?

8 В чём заключается сущность метода Чепина?

9 Какие типы переменных используются в метрике Чепина?

10 Как рассчитывается метрика Чепина?

11 Какие параметры определяют структурную сложность программ?

12 В чём заключается сущность метрики структурной сложности программ?

13 Сколько и какие критерии оценки структурной сложности программ используются для оценки качества программного обеспечения?

14 В чём заключается сущность первого критерия структурной сложности программ?

15 В чём заключается сущность второго критерия структурной сложности программ?

16 В чём заключается сущность третьего критерия структурной сложности программ?

17 Что характеризует метрика Маккейба?

18 Какие этапы включает процесс построения графа потока управления?

19 Какие существуют показатели оценки качества процедурно-ориентированных программных средств?

20 Как рассчитывается оценка качества процедурно-ориентированных программных средств?

21 Какие виды косвенных метрик применяются для оценки качества процедурно-ориентированных программных средств?

22 Что называется связностью программных модулей?

23 Какие существуют типы связностей программных модулей?

24 Что называется сцеплением программных модулей?

25 Какие метрики сцепления программных модулей используются для оценки программных средств?

26 Сколько и какие объектно-ориентированные метрики применяют для оценки качества программных средств?

27 Сколько и какие метрики Мартина применяют на практике?

28 Сколько и какие метрики Чидамбера и Кемерера применяют на практике?

29 Сколько и какие метрики Лоренца и Кидда применяют на практике?

30 Какие показатели качества включает набор метрик Абреу?

Задания для самостоятельной работы

1 Разработайте программу на языке C++ решения нелинейного уравнения. Дано действительное положительное число ε. Методом хорд вычислить с точностью ε корень уравнения $x * 2^x - 1 = 0$, [0; 1]. В квадратных скобках указан отрезок, содержащий корень.

После реализации программного алгоритма выполните следующие действия:

- сформируйте словарь программы, охватывающий операнды, операторы и операции (таблицы 2.2-2.5);
- рассчитайте метрики Холстеда, оформив результат в виде итоговой таблицы (таблица 2.5);
- проведите анализ полученных результатов и сделайте вывод.

Таблица 2.2 – Словарь операторов и операций программы

№ п/п	Операторы, операции	Номера строк	Количество повторений

Таблица 2.3 – Словарь операндов программы

№ п/п	Операнды	Номера строк	Количество повторений

Таблица 2.4 – Входные и выходные переменные программы

Входные переменные	Выходные переменные

Таблица 2.5 – Значения метрик Холстеда для программы

Наименование характеристики	Обозначение, формула для вычисления	Значение

2 Разработайте программу на языке С++ вычисления значений функции и на основе лексического анализа исходного текста программы оцените качество с использованием метрик Джилба (рисунок 2.1).

3 Дана целочисленная прямоугольная матрица $[A_{n*m}]$. Разработайте программу на языке С++, определяющую сумму элементов матрицы, находящихся ниже побочной диагонали. Элементы матрицы генерируются датчиком случайных чисел. На основе лексического анализа исходного текста программы определите значение метрики Чепина.

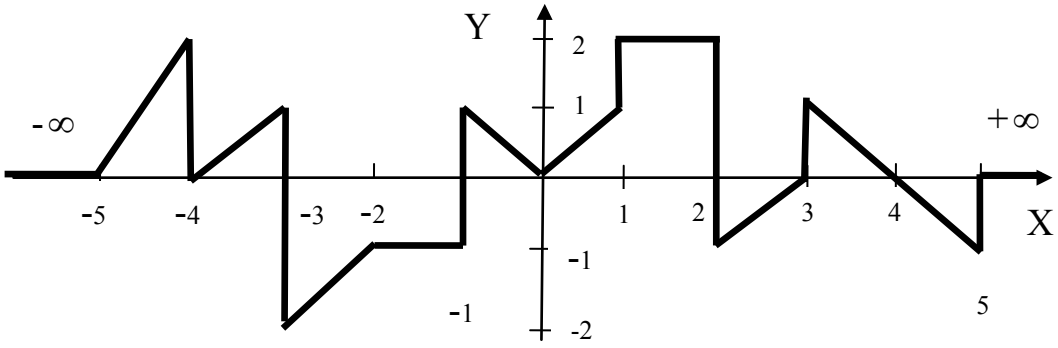


Рисунок 2.1 – Графический способ задания функций

4 Разработайте программу на языке С++ определения попадания точки с введёнными координатами в закрашенную область (рисунок 2.2).

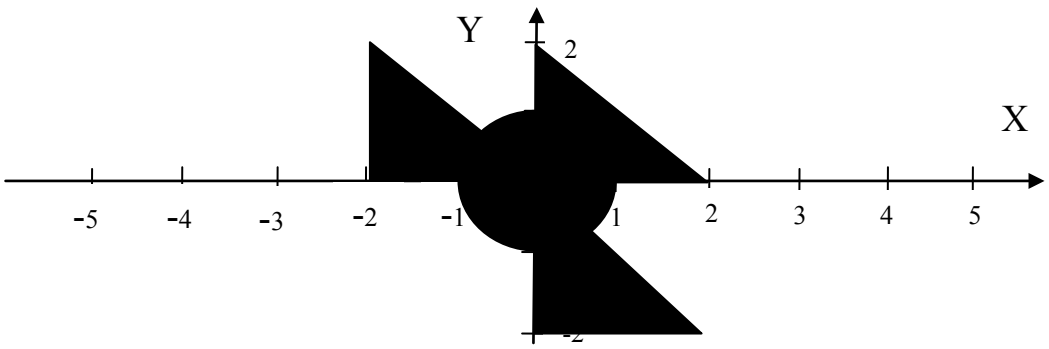


Рисунок 2.2 – Вид фигуры на плоскости

Оцените алгоритмическую сложность программы:

- постройте граф потока управления;
- сформируйте маршруты тестирования в соответствии с критериями 1, 2, 3;
- определите значение цикломатического числа, характеризующего структурную сложность программ;
- сформируйте матрицы смежности и достижимости;
- проведите анализ результатов и сделайте выводы.

5 Дана целочисленная прямоугольная матрица A_{n*m}. Разработайте программу на языке C++, определяющую средние квадратичные значения четных столбцов и нечетных строк матрицы. После программной реализации алгоритма выполните действия:

- рассчитайте функциональные указатели;
- оцените уровень связности и силу сцепления программных модулей;
- проведите анализ полученных результатов и сделайте выводы.

6 Разработайте программу на языке C++. Оцените характеристики программы на основе применения объектно-ориентированных метрик Мартина. Создайте базовый класс **Car** (машина), характеризуемый торговой маркой (строка), числом цилиндров, мощностью. Определите методы переназначения и изменения мощности. Создайте производный класс Lorry (грузовик), характеризуемый также грузоподъемностью кузова. Определите функции переназначения марки и изменения грузоподъемности.

7 Разработайте программу на языке C++. Оцените характеристики программы на основе применения объектно-ориентированных метрик Чидамбера и Кемерера. Создайте класс **Payment** (зарплата). В классе должны быть представлены поля: фамилия-имя-отчество, оклад, год поступления на работу, процент надбавки, подоходный налог, количество отработанных дней в месяце, количество рабочих дней в месяце, начисленная и удержанная суммы. Реализовать методы:

- вычисления начисленной суммы;

- вычисления удержанной суммы;
- вычисления суммы, выдаваемой на руки;
- вычисления стажа.

Стаж вычисляется как полное количество лет, прошедших от года поступления на работу до текущего года. Начисления представляют собой сумму, начисленную за отработанные дни, и надбавки, то есть доли от первой суммы. Удержания представляют собой отчисления в пенсионный фонд (1% от начисленной суммы) и подоходный налог. Подоходный налог составляет 13% от начисленной суммы без отчислений в пенсионный фонд.

8 Разработайте программу на языке C++. Оцените характеристики программы на основе применения объектно-ориентированных метрик Лоренца и Кидда. Создайте класс **Bill** (счет), представляющий собой разовый платеж за телефонный разговор. Класс должен включать в себя поля: номер телефона, тариф за минуту разговора, скидка (в процентах), время разговора (в минутах) и сумма к оплате. Реализуйте метод вычисления суммы к оплате. В программе продемонстрировать создание, инициализацию и обработку массива объектов типа **Bill** с различными исходными данными для вычисления сумм к оплате. Вычислите общую сумму к оплате.

9 Разработайте программу на языке C++. Оцените характеристики программы на основе применения объектно-ориентированных метрик Абреу. Создайте класс **Point** (точка), который имеет поля – координаты точки, класс **Ellipse** (эллипс) и класс **Circle (**окружность). Определите иерархию типов. Определите функции печати, конструкторы, деструкторы, вычисление площади.

3 Обеспечение надёжности разработки программных приложений

Предметом изучения теории надёжности комплексов программ (Software Reliability) является работоспособность сложных программ обработки информации в реальном времени.

Задачи теории и анализа надёжности сложных программных средств:

1 формулирование основных понятий, используемых при исследовании и применении показателей надёжности программных средств;

2 выявление и исследование основных факторов, определяющих характеристики надёжности сложных программных комплексов;

3 выбор и обоснование критериев надёжности для комплексов программ различного типа и назначения;

4 исследование дефектов и ошибок, динамики изменения при отладке и сопровождении, а также влияния на показатели надёжности программных средств;

5 исследование и разработка методов структурного построения сложных программных средств, обеспечивающих необходимую надёжность;

6 исследование методов и средств контроля и защиты от искажений программ, вычислительного процесса и данных путём использования различных видов избыточности и помехозащиты;

7 разработка методов и средств определения и прогнозирования характеристик надёжности в жизненном цикле комплексов программ с учётом функционального назначения, сложности, структурного построения и технологии разработки.

Решение задач позволяет создать программные средства с заданными показателями надёжности. В жизненном цикле программных средств значения показателей качества и надёжности компонентов и комплексов программ необходимо анализировать и прогнозировать с целью обеспечения заданных показателей надёжности [6].

3.1 Базовые показатели надёжности программного обеспечения

Оценка надёжности программного обеспечения заключается в определении вероятности безотказной работы программы в заданных условиях в течение определённого периода времени.

Факторы, влияющие на оценку надёжности:

- программные методы и средства технологии программирования, применяемые в процессах разработки и способствующие достижению требуемой надёжности;
- тестирование и проверка функционирования созданного программного средства со сбором данных о результатах обнаружения ошибок и интенсивности отказов в интервалах времени функционирования.

Показатели надёжности программного обеспечения:

1 функция надёжности $P(t)$ – вероятность того, что ни одна ошибка не появится на интервале от 0 до t;

2 функция отказов Q(t) – вероятность того, что ошибка появится на интервале от 0 до t;

3 $f(t)$ – плотность распределения времени безотказной работы программного обеспечения;

4 функция риска $\lambda(t)$ – интенсивность отказов, или условная вероятность того, что ошибка появится на интервале от 0 до $t + \Delta t$ при условии, что на интервале от 0 до t ошибок не было.

5 T_{cp} – среднее время между отказами [5].

Функция надёжности $P(t)$ описывается математическим выражением

$$P(t) = P(T \geq t), \qquad (3.1)$$

где t – время.

Функция отказов $Q(t)$ описывается математическим выражением

$$Q(t) = 1 - P(t). \qquad (3.2)$$

Плотность распределения времени безотказной работы программного обеспечения представляется математическим выражением

$$f(t) = Q'(t) = -P'(t). \qquad (3.3)$$

Функция риска $\lambda(t)$ описывается математическим выражением

$$\lambda(t) = -\frac{P'(t)}{P(t)} = \frac{f(t)}{P(t)}, \qquad (3.4)$$

где $P(t)$ – вероятность того, что ни одна ошибка не появится на интервале от 0 до t.

Функция надёжности определяется по формуле

$$P(t) = \exp\left(-\int_0^t \lambda(u)du\right). \qquad (3.5)$$

Среднее время между отказами определяется по формуле [5]

$$T_{cp} = \int_0^{+\infty} P(u)du = \int_0^{+\infty} u * f(u)du. \qquad (3.6)$$

3.2 Классификация моделей надёжности программного обеспечения

Модель надёжности программного обеспечения – математическая модель, построенная для оценки зависимости надёжности программного обеспечения от параметров.

Классификация моделей надёжности программного обеспечения приведена на рисунке 3.1.

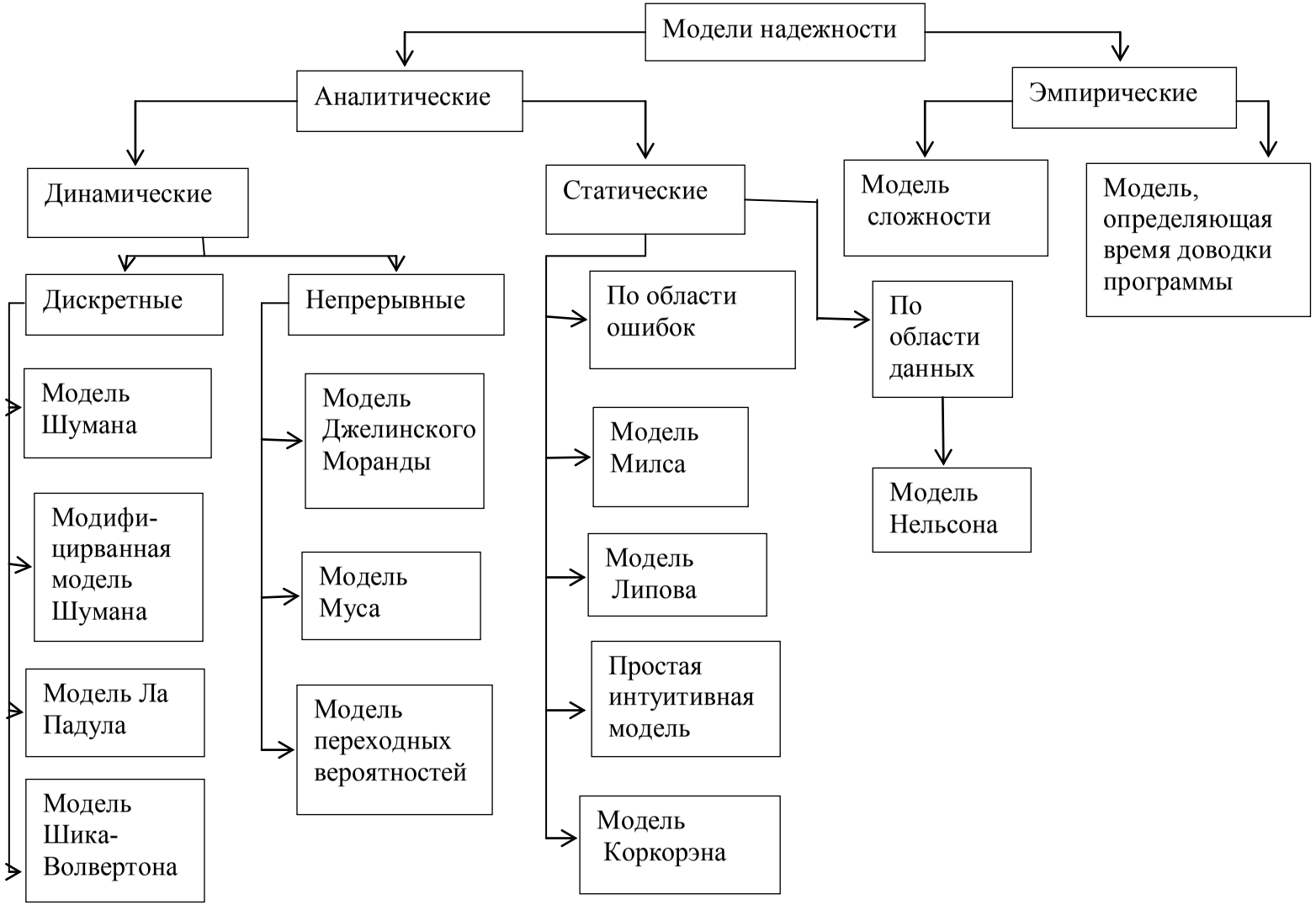


Рисунок 3.1 – Классификационная схема моделей надежности программного обеспечения

Значения параметров предполагаются известными либо могут быть измерены в ходе наблюдений или экспериментального исследования процесса функционирования программного обеспечения [6].

Модели надёжности программных средств подразделяются на аналитические и эмпирические. Аналитические модели дают возможность рассчитать количественные показатели надёжности, основываясь на данных о поведении программы в процессе тестирования (измеряющие и оценивающие модели). Эмпирические модели основываются на анализе структурных особенностей программ. Они рассматривают зависимость показателей надёжности от числа межмодульных связей, количества циклов в модулях, отношения количества прямолинейных участков программы к количеству точек ветвления [6].

Эмпирические модели не дают конечных результатов показателей надёжности. Они позволяют выявлять взаимосвязь между сложностью программных средств и надёжностью. Эмпирические модели используют на этапе проектирования программных средств, когда осуществлена разбивка на модули и известна структура.

Аналитические модели представлены двумя группами:

- динамические модели;
- статические модели.

В динамических моделях надёжности программных средств поведение программных средств (появление отказов) рассматривается во времени. В статических моделях надёжности программных средств появление отказов не связывают со временем, а учитывают зависимость количества ошибок от числа тестовых прогонов (по области ошибок) или зависимость количества ошибок от характеристики входных данных (по области данных).

Для применения динамических моделей надёжности программных средств необходимы данные о появлении отказов во времени. Если фиксируются интервалы каждого отказа, то получается непрерывное появление отказов во времени (динамические модели с непрерывным временем). В случае появления отказов в дискретных точках применяются динамические модели с дискретным временем [6].

Аналитическое моделирование надёжности программных средств включает четыре шага:

1 определение предположений, связанных с процедурой тестирования программных средств;

2 разработку или выбор аналитической модели, базирующейся на предположениях о процедуре тестирования;

3 выбор параметров моделей с использованием полученных данных;

4 применение модели – расчёт количественных показателей надёжности по модели [6].

3.3 Аналитические динамические модели надёжности программного обеспечения

3.3.1 Дискретные модели надёжности программного обеспечения

В дискретных динамических моделях поведение программного обеспечения рассматривается в дискретных точках: число отказов фиксируется за произвольный интервал времени.

3.3.1.1 Модель Шумана

Исходные данные для модели Шумана собираются во время тестирования программы в течение фиксированных или случайных временных интервалов. Каждый интервал – стадия, на которой выполняется последовательность тестов и фиксируется число ошибок. Тестирование проводится в несколько этапов. Каждый этап – выполнение программы на полном комплексе разработанных тестовых данных. Вычисленные ошибки регистрируются, но не исправляются. Создаются исходные статистические данные об ошибках. После завершения этапа на основе собранных данных используется модель Шумана для расчёта количественных показателей надёжности. После этого исправляются ошибки, обнаруженные на предыдущем этапе, корректируются тестовые наборы и проводится новый этап тестирования.

Исходное количество ошибок в программе постоянно и в процессе тестирования может уменьшаться по мере выявления и исправления ошибок. Новые ошибки при корректировке не вносятся. Скорость обнаружения ошибок пропорциональна числу оставшихся ошибок. Общее число машинных инструкций в рамках одного этапа тестирования постоянно.

Пусть до начала тестирования в программе имеется E_T ошибок. При тестировании за время τ обнаруживается ε_C ошибок в расчёте на команду в машинном языке.

Удельное число ошибок на одну машинную команду, оставшихся в системе после τ времени тестирования, определяется по формуле

$$\varepsilon_r(\tau) = \frac{E_T}{I_T} * \varepsilon_C(\tau), \qquad (3.7)$$

где I_T – общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования;

E_T – количество ошибок до начала тестирования программы.

Пусть $Z(t)$ – функция частоты отказов, пропорциональна числу ошибок, оставшихся в программе после времени тестирования τ.

Функция отказов описывается математическим выражением

$$Z(t) = C * \varepsilon_r(\tau), \tag{3.8}$$

где C – коэффициент пропорциональности;

$\varepsilon_r(\tau)$ – удельное число ошибок на одну машинную команду, оставшихся в системе после τ времени тестирования;

t – время работы программы без отказа.

Если время работы программы без отказа t отсчитывается от $t = 0$, а τ остаётся фиксированным, то вероятность безотказной работы на интервале времени от 0 до t (функция надёжности) определяется по формуле

$$P(t, \tau) = \exp\{C * [\frac{E_T}{I_T} - \varepsilon_C(\tau)] * t\}, \tag{3.9}$$

где C – коэффициент пропорциональности;

E_T – количество ошибок до начала тестирования программы;

I_T – общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования;

ε_C – количество ошибок в расчёте на команду в машинном языке, обнаруженных при тестировании за время τ;

t – время работы программы без отказа.

Средне время определяется по формуле

$$t_{cp} = \frac{1}{C * [\frac{E_T}{I_T} - \varepsilon_C(\tau)]}, \tag{3.10}$$

где C – коэффициент пропорциональности;

E_T – количество ошибок до начала тестирования программы;

I_T – общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования;

ε_C – количество ошибок в расчёте на команду в машинном языке, обнаруженных при тестировании за время τ.

В формулах 3.8 и 3.9 неизвестными параметрами являются коэффициент пропорциональности C и количество ошибок до начала тестирования программы E_T.

Количество ошибок до начала тестирования программы E_T определяется по формуле

$$E_T = \frac{I_T * [\lambda_{\tau_a} * \varepsilon_c(\tau_a) - \lambda_{\tau_b} * \varepsilon_c(\tau_b)]}{\lambda_{\tau_a} - \lambda_{\tau_b}}, \quad (3.11)$$

где I_T – общее число машинных команд;

$\varepsilon_c(\tau_a)$ – количество ошибок в расчёте на команду в машинном языке, обнаруженных в момент тестирования τ_a;

$\varepsilon_c(\tau_b)$ – количество ошибок в расчёте на команду в машинном языке, обнаруженных в момент тестирования τ_b;

λ_{τ_a} – интенсивность появления ошибок в момент тестирования τ_a;

λ_{τ_b} – интенсивность появления ошибок в момент тестирования τ_b.

Коэффициент пропорциональности C определяется по формуле

$$C = \frac{\lambda_{\tau_a}}{\frac{E_T}{I_T} - \varepsilon_c(\tau_a)}, \quad (3.12)$$

где E_T – количество ошибок до начала тестирования программы;

I_T – общее число машинных команд;

$\varepsilon_c(\tau_a)$ – количество ошибок в расчёте на команду в машинном языке, обнаруженных в момент тестирования τ_a;

λ_{τ_a} – интенсивность появления ошибок в момент тестирования τ_a.

В процессе тестирования программы собирается информация о времени и количестве ошибок на каждом прогоне программы.

Общее время тестирования τ определяется по формуле

$$\tau = \sum_{i=1}^{n} \tau_i, \quad (3.13)$$

где τ_i – время i-го прогона программы, $i = \overline{1,n}$;

n – количество прогонов программы;

i – номер прогона программы.

Интенсивность появления ошибок λ постоянна. Интенсивность появления ошибок определяется по формуле

$$\lambda = \frac{\sum_{i=1}^{n} A_i}{\tau}, \qquad (3.14)$$

где A_i – количество ошибок в i-м прогоне программы, $i = \overline{1,n}$;

n – количество прогонов программы;

i – номер прогона программы.

Среднее время определяется по формуле

$$t_{cp} = \frac{\tau}{\sum_{i=1}^{n} A_i}, \qquad (3.15)$$

где A_i – количество ошибок в i-м прогоне программы, $i = \overline{1,n}$;

n – количество прогонов программы;

i – номер прогона программы;

τ – общее время тестирования программы.

Пусть τ_a и τ_b – два различных момента тестирования, которые выбираются произвольно с учётом требования $\varepsilon_c(\tau_b) > \varepsilon_c(\tau_a)$.

Сопоставляя уравнения 3.10 и 3.15 в моменты времени тестирования программы τ_a и τ_b, получим формулы расчёта величин $\frac{1}{\lambda_{\tau_a}}$ и $\frac{1}{\lambda_{\tau_b}}$.

Величина $\frac{1}{\lambda_{\tau_a}}$ определяется по формуле

$$\frac{1}{\lambda_{\tau_a}} = \frac{1}{C * [\frac{E_T}{I_T} - \varepsilon_c(\tau_a)]}, \qquad (3.16)$$

где C – коэффициент пропорциональности;

E_T – количество ошибок до начала тестирования программы;

I_T – общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования;

$\varepsilon_c(\tau_a)$ – количество ошибок в расчёте на команду в машинном языке, обнаруженных в момент тестирования τ_a;

λ_{τ_a} – интенсивность появления ошибок в момент тестирования τ_a.

Величина $\frac{1}{\lambda_{\tau b}}$ определяется по формуле

$$\frac{1}{\lambda_{\tau b}} = \frac{1}{C * [\frac{E_T}{I_T} - \varepsilon_c(\tau_b)]}, \quad (3.17)$$

где C – коэффициент пропорциональности;

E_T – количество ошибок до начала тестирования программы;

I_T – общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования;

$\varepsilon_c(\tau_b)$ – количество ошибок в расчёте на команду в машинном языке, обнаруженных в момент тестирования τ_b;

$\lambda_{\tau b}$ – интенсивность появления ошибок в момент тестирования τ_b.

Преимущество модели Шумана – возможность определения всех неизвестных параметров, что сокращает время расчёта надёжности.

Недостатки модели Шумана:

- регистрация большого количества данных, необходимых для расчёта;
- при корректировке не вносятся новые ошибки, что не всегда имеет место в реальных программах.

3.3.1.2 Модель Ла Падула

Модель Ла-Падула предусматривает выполнение тестов в m этапов, каждый из которых заканчивается внесением изменений (исправлений) в программное обеспечение.

Надёжность программного обеспечения определяется по формуле

$$R(t) = R(\infty) - \frac{A}{i}, \quad (3.18)$$

где A – параметр роста;

$R(\infty)$ – предельная надёжность программного обеспечения;

i – номер этапа, $i = \overline{1, m}$;

m – количество этапов.

Предельная надёжность программного обеспечения определяется по формуле

$$R(\infty) = \lim_{i \to \infty} R(i), \tag{3.19}$$

Неизвестные параметры определяются решением уравнений

$$\begin{cases} \sum_{i=1}^{m} \left(\frac{S_i - m_i}{S_i} - R(\infty) + \frac{A_i}{i}\right) = 0 \\ \sum_{i=1}^{m} \left(\frac{S_i - m_i}{S_i} - R(\infty) + \frac{A_i}{i}\right) * \frac{1}{i} = 0 \end{cases}, \tag{3.20}$$

где S_i – число тестов;

m_i – число отказов во время i-го этапа;

i – номер этапа, $i = \overline{1, m}$;

m – количество этапов.

Преимущество модели Ла-Падула – прогнозная модель, предсказывающая вероятность безотказной работы программного приложения на последующих этапах выполнения [5].

3.3.2 Непрерывные модели надёжности программного обеспечения

Непрерывные динамические модели надёжности программного обеспечения определяют интервалы времени каждого отказа с получением непрерывного появления отказов во времени.

3.3.2.1 Модель Джелинского-Моранды

В модели Джелинского-Моранды значение интервалов времени тестирования между обнаружением двух ошибок имеет экспоненциальное распределение с интенсивностью отказов, пропорциональной числу не выявленных ошибок. Обнаруженная ошибка устраняется, число оставшихся ошибок уменьшается на единицу.

Функция плотности распределения времени обнаружения i-й ошибки, отсчитываемого от момента выявления $(i-1)$-й ошибки, имеет вид

$$f(t_i) = \lambda_i * \exp(-\lambda_i * t_i), \tag{3.21}$$

где λ_i – интенсивность отказов;

t_i –время, $i = \overline{1, n}$.

Вероятность безотказной работы программного обеспечения определяется по формуле

$$P(t_i) = \exp(-\lambda_i * t_i), \tag{3.22}$$

где λ_i – интенсивность отказов;

t_i – время, $i = \overline{1,n}$.

Интенсивность отказов, пропорциональная числу невыявленных ошибок в программе, определяется по формуле

$$\lambda_i = C * (N - i + 1), \tag{3.23}$$

где C – коэффициент пропорциональности;

N – первоначальное количество ошибок программы.

Наиболее вероятные значения параметров C и N определяются при тестировании программы на основе полученных данных. Для этого определяется время выполнения программы до очередного отказа $t_1, t_2, ...t_n$ и решается система уравнений

$$\begin{cases} \sum\limits_{i=1}^{n} \dfrac{1}{(N-i+1)} = \dfrac{K}{N+1-Q*K} \\ C = \dfrac{K}{A*(N+1-Q*K)} \end{cases}, \tag{3.24}$$

где Q – параметр, определяемый по формуле $Q = \dfrac{B}{A*K}$;

A – параметр, определяемый по формуле $A = \sum\limits_{i=1}^{n} t_i$;

B – параметр, определяемый по формуле $B = \sum\limits_{i=1}^{n} i * t_i$.

Преимуществом модели Джелинского-Моранды является простота расчётов.

Недостатки модели Джелинского-Моранды:

- при неточном определении величины N интенсивность отказов программы может стать отрицательной;
- при исправлении обнаруженных ошибок не вносятся новые ошибки [5].

3.3.2.2 Модель Муса

Модель Муса предполагает, что в процессе тестирования фиксируется время выполнения программы до очередного отказа. Не всякая ошибка программного обеспечения может вызвать отказ, поэтому допускается обнаружение более одной ошибки при выполнении программы до возникновения очередного отказа.

Пусть M_0 – количество отказов на протяжении всего жизненного цикла программного обеспечения, N_0 – количество ошибок в программе до начала тестирования. Функция зависимости количества ошибок в программе до начала тестирования от количества отказов на протяжении всего жизненного цикла программного обеспечения имеет вид

$$N_0 = B * M_0, \tag{3.25}$$

где B – коэффициент уменьшения количества ошибок;

M_0 – количество отказов на протяжении всего жизненного цикла программного обеспечения.

Коэффициент уменьшения числа ошибок определяется по формуле

$$B = \frac{n}{m}, \tag{3.26}$$

где n – число выявленных ошибок;

m – число отказов, зафиксированных во время тестирования.

Виды времени модели Муса:

- суммарное время функционирования;
- оперативное время выполнения программы.

Суммарное время функционирования – время тестирования до контрольного момента, когда проводится оценка надёжности.

Оперативное время выполнения программы – время безотказной работы на этапе эксплуатации (время от контрольного момента и далее при условии, что дальнейшего устранения ошибок не будет).

Интенсивность отказов для суммарного времени функционирования пропорциональна количеству неустранённых ошибок. Скорость изменения числа устранённых ошибок, измеряемая относительно суммарного времени, пропорциональна интенсивности отказов.

Показатели надёжности модели Муса:

- средняя наработка на отказ;
- среднее число отказов.

Средняя наработка на отказ рассчитывается как математическое ожидание временного интервала между последовательными отказами и определяется по формулам

$$T_{cp} = \int_0^{+\infty} P(t)dt; \tag{3.27}$$

$$T_{cp} = T_0 * \exp(\frac{c * \tau}{M_0 * T_0}), \tag{3.28}$$

где T_0 – начальная наработка на отказ;

τ – время функционирования;

c – коэффициент сжатия тестов (равен времени испытаний).

Среднее число отказов определяется по формуле

$$m = M_0 * (1 - \exp(\frac{c * \tau}{M_0 * T_0})), \tag{3.29}$$

где T_0 – начальная наработка на отказ;

τ – время функционирования;

c – коэффициент сжатия тестов (равен времени испытаний);

M_0 – количество отказов на протяжении всего жизненного цикла программного обеспечения.

Если интенсивность отказов постоянна (длительность интервалов между последовательными отказами имеет экспоненциальное распределение), то средняя наработка на отказ обратно пропорциональна интенсивности отказов.

Преимущество модели Муса – отсутствие фиксации моментов отказов. В случае появления отказов ошибки регистрируются, а исправляются по завершении этапа тестирования.

Недостаток модели Муса – дополнительные затраты времени. Для определения первоначального числа ошибок в программном обеспечении необходимо проведение расчётов по другой модели [5].

3.4 Аналитические статические модели надёжности программного обеспечения

Статические модели надёжности программного обеспечения отличаются от динамических моделей отсутствием времени появления ошибок

в процессе тестирования и неиспользованием предположений о поведении функции риска. Они строятся на статистическом фундаменте [6].

3.4.1 Модели надёжности программного обеспечения по области ошибок

3.4.1.1 Модель Миллса

Статическая модель надёжности программного обеспечения позволяет оценить количество ошибок до начала тестирования и степень отлаженности программы. Для применения модели Миллса в программу вносятся ошибки и предполагают равновероятное обнаружение внесённых и собственных ошибок программы. Ошибки вносятся случайным образом и учитываются в протоколе ошибок. Количество и характер ошибок тестировщику неизвестны.

В течение определённого времени программа тестируется, и определяются статистические данные об обнаруженных в программе ошибках [5].

Оценка количества ошибок до начала тестирования определяется по формуле

$$N = \frac{W * S}{V}, \tag{3.30}$$

где W – количество внесённых в программу ошибок;

V – количество обнаруженных в процессе тестирования ошибок из числа внесённых;

S – количество собственных ошибок программы.

Степень отлаженности программы определяется по формуле

$$C = \begin{cases} 1, \text{если S} > \text{r} \\ \dfrac{W}{W + r + 1}, \text{если } S \le r \end{cases}, \tag{3.31}$$

где W – количество внесённых в программу ошибок;

V – количество обнаруженных в процессе тестирования ошибок из числа внесённых;

S – количество собственных ошибок программы;

r – верхний предел (максимум) предполагаемого количества собственных ошибок в программе.

Выражения 3.30 и 3.31 представляют модель Миллса.

Если обнаружено V ошибок из внесённых W, применяется математическое выражение

$$C = \begin{cases} 1, \text{если S} > \text{r} \\ \dfrac{\binom{S}{V-1}}{\binom{S+r+1}{r+V}}, \text{если } S \le r \end{cases}, \tag{3.32}$$

где $\binom{S}{V-1}$ – число сочетаний из S элементов по $V-1$ элементов в каждой комбинации;

$\binom{S+r+1}{r+V}$ – число сочетаний из $S+r+1$ элементов по $r+V$ элементов в каждой комбинации.

Преимущества модели Миллса:

- простота математического аппарата;
- наглядность.

Недостатки модели Миллса:

- необходимость внесения искусственных ошибок;
- экспертное оценивание коэффициента r [12].

3.4.1.2 Модель Липова

Модель Липова является модифицированной моделью Миллса. Собственные и внесённые ошибки программы имеют равную вероятность обнаружения. Модель Липова позволяет оценить вероятность обнаружения определённого количества ошибок к моменту оценки.

Вероятность обнаружения собственных и внесённых ошибок определяется по формуле

$$Q(n,V) = (\frac{m}{n+V}) * q^{n+V} * (1-q)^{m-n-V} * \frac{\frac{N}{n} * \frac{S}{V}}{\frac{N+S}{n+V}}, \tag{3.33}$$

где n – количество собственных ошибок;

V – количество внесённых ошибок;

m – количество тестов, используемых при тестировании программы;

q – вероятность обнаружения ошибки в каждом из m тестов;

S – общее количество искусственно внесённых ошибок;

N – количество собственных ошибок, имеющихся в программном обеспечении до начала тестирования.

Вероятность обнаружения ошибки в каждом из m тестов определяется по формуле

$$q = \frac{n+V}{n}, \tag{3.34}$$

где n – количество собственных ошибок;

V – количество внесённых ошибок.

Для использования модели Липова необходимо выполнение условий:

$$N \geq n \geq 0; S \geq V \geq 0; m \geq n + V \geq 0. \tag{3.35}$$

Оценки максимального правдоподобия N задаются математическими выражениями

$$N = \frac{S * n}{V}, \text{ при } n \geq 1,\ V \geq 1, \tag{3.36}$$

$$N = n * S, \text{ при } V = 0, \tag{3.37}$$

$$N = 0, \text{ при } n = 0, \tag{3.38}$$

где n – количество собственных ошибок;

V – количество внесённых ошибок;

m – количество тестов, используемых при тестировании программы;

S – общее количество искусственно внесённых ошибок;

N – количество собственных ошибок, имеющихся в программном обеспечении до начала тестирования [6].

3.4.1.3 Модель Коркорэна

Модель Коркорэна относится к статическим моделям надёжности программного обеспечения, т.к. не использует параметры времени и учитывает только результат. Модель Коркорэна предполагает разную вероятность появления ошибок в программе. Оценка надёжности программного обеспечения определяется по формуле

$$P = \frac{M_0}{M} + \sum_{i=1}^{k} \frac{\delta_i \ (n_i - 1)}{M}, \qquad (3.39)$$

где M_0 – число успешных прогонов программного обеспечения;

M – общее число прогонов;

k – количество типов ошибок;

δ_i – коэффициент;

Коэффициент δ_i определяется по формуле

$$\delta_i = \begin{cases} p, \text{если } n_i > 0 \\ 0, \text{если } n_i = 0 \end{cases}, \qquad (3.40)$$

где p_i – вероятность ошибки i-го типа при тестировании.

Преимущества модели Коркорэна:

- учёт нескольких источников ошибок в программном обеспечении;
- простой математический расчёт надёжности программного обеспечения.

Недостаток модели Коркорэна – необходимость определения статистическим методом вероятности выбора набора данных из предполагаемой области при прогоне программы [5].

3.4.2 Модели надёжности программного обеспечения по области данных

3.4.2.1 Модель Нельсона

Модель Нельсона основана на выделении областей исходных данных E_i, покрывающих всё множество вариантов использования в программе E.

Множество вариантов использования в программе представляется выражением

$$E = \{E_i\}. \qquad (3.41)$$

Множество результатов выполнения программы $F(E)$ определяется по формуле

$$F(E) = \bigcup_{I} F(E_i). \qquad (3.42)$$

Пусть $F_{\psi}(E_i)$ – множество фактических результатов выполнения программы на наборе E_i.

Множество правильных значений определяется по формуле

$$\forall i \left| F(E_i - F_{\psi}(E_i) \right| \le \varepsilon_i, \qquad (3.43)$$

где ε_i – допустимое расхождение между правильным и фактическим результатом выполнения программы на наборе E_i.

При выполнении неравенства 3.44 наступает рабочий отказ – ситуация бесконечного выполнения программы

$$\forall i \left| F(E_i - F_{\psi}(E_i) \right| > \varepsilon_i. \qquad (3.44)$$

Пусть N – мощность множества набора исходных данных E, E_0 – множество, состоящее из всех наборов E_i, для которых получены неудовлетворительные результаты, N_0 – мощность множества E_0.

Прогон программы – процесс, включающий ввод E_i и выполнение программы, который заканчивается получением результата $F_{\psi}(E_i)$ или рабочим отказом.

Вероятность P рабочего отказа при прогонке программы равна вероятности того, что набор данных E_i, который использовался в прогоне программы, принадлежит множеству E_0.

Вероятность появления ошибки при прогоне программы на входном наборе, случайно выбранном из числа равновероятных, определяется по формуле

$$P = \frac{N_0}{N}, \qquad (3.45)$$

где N – мощность множества набора исходных данных E;

N_0 – мощность множества E_0.

Вероятность R приемлемого результата прогона программы на наборе входных данных E_i, случайно выбранном из E среди равновероятных наборов, определяется по формуле

$$R = 1 - P = 1 - \frac{N_0}{N}, \qquad (3.46)$$

где N – мощность множества набора исходных данных E;

N_0 – мощность множества E_0.

P – вероятность появления ошибки при прогоне программы на входном наборе, случайно выбранном из числа равновероятных.

Оценка надёжности программы при неравновероятном наборе данных из множества E определяется по формуле

$$R = 1 - \sum_{i=1}^{N} p_i * y_i, \qquad (3.47)$$

где p_i – вероятность использования i-го набора исходных данных;

y_i – динамическая переменная, которая принимает нулевое значение, если прогон заканчивается приемлемым результатом, и значение 1, если прогон заканчивается рабочим отказом.

Вероятность $R(n)$ успешного выполнения n прогонов программы при независимом для каждого прогона выборе исходных данных определяется выражением

$$R(n) = e^{\sum_{j=1}^{n} \ln(1-p_j)}, \qquad (3.48)$$

где p_j – вероятность отказа для j-го прогона;

Преимущество модели Нельсона – расчёт надёжности на всех этапах жизненного цикла программного обеспечения [5; 12].

3.5 Эмпирические модели надёжности программного обеспечения

Эмпирические модели надёжности программных средств основаны на анализе структурных особенностей программных средств. Они используются для прогнозирования требующихся ресурсов тестирования, уточнения плановых сроков завершения проекта на этапе проектирования программных средств [6].

Преимущества эмпирических моделей:

- отсутствие сложных формул;
- простые вычисления.

Недостатки эмпирических моделей:

- приблизительные расчёты;
- отсутствие отражения динамики вычислительного процесса при эксплуатации программ [5].

3.5.1 Модель фирмы IBM

Эмпирическая модель фирмы IBM оценивает число ошибок по формуле

$$N_0 = 23 * M_{10} + 2 * M_1, \tag{3.49}$$

где M_{10} – число модулей, потребовавших 10 и более исправлений;

M_1 – число модулей, в которых обнаружено меньше 10 ошибок.

Оценка средней наработки программного обеспечения на отказ в часах определяется по формуле

$$T_{cp} = \alpha * \frac{V_{op}}{N_0}, \tag{3.50}$$

где V_{op} – объём программы в операторах;

N_0 – число ошибок в программном обеспечении;

α – коэффициент, лежащий от 100 до 1000 [5].

3.5.2 Модель Холстеда

Модель Холстеда оценивает количество оставшихся в программе ошибок после окончания разработки. Количество оставшихся ошибок в программе определяется по формуле

$$N_0 = K_{HO} * V_{ОП} * \log_2(\eta_1 + \eta_2), \tag{3.51}$$

где V_{HO} – коэффициент пропорциональности;

$V_{ОП}$ – число операторов в программе;

η_1 – число операторов в программном средстве;

η_2 – число операндов в программном средстве [5].

Контрольные вопросы

1 Что является предметом изучения теории надёжности комплексов программ?

2 Какие задачи решает теория надёжности комплексов программ?

3 В чём заключается сущность оценки надёжности программного обеспечения?

4 Какие факторы влияют на оценку надёжности программного обеспечения?

5 Какие показатели надёжности применяются для оценки программного обеспечения?

6 Что понимается под моделью надёжности программного обеспечения?

7 Каким образом классифицируются модели надёжности программного обеспечения?

8 Какие шаги включает аналитическое моделирование надёжности программных средств?

9 В чём заключается сущность динамических моделей надёжности программного обеспечения?

10 В чём заключается сущность статических моделей надёжности программного обеспечения?

11 В чём заключается сущность эмпирических моделей надёжности программного обеспечения?

12 Какие виды дискретных моделей применяются при оценке надёжности программного обеспечения?

13 Какие существуют особенности модели Шумана? Какие преимущества и недостатки имеет модель Шумана?

14 Какие существуют особенности модели Ла Падула? Какие преимущества и недостатки имеет модель Ла Падула?

15 Какие виды непрерывных моделей применяются при оценке надёжности программного обеспечения?

16 Какие существуют особенности модели Джелинского-Моранды? Какие преимущества и недостатки имеет модель Джелинского-Моранды?

17 Какие существуют особенности модели Муса? Какие преимущества и недостатки имеет модель Муса?

18 Какие виды статических моделей применяются при оценке надёжности программного обеспечения?

19 Какие существуют особенности модели Миллса? Какие преимущества и недостатки имеет модель Миллса?

20 Какие существуют особенности модели Липова? Какие преимущества и недостатки имеет модель Липова?

21 Какие существуют особенности модели Коркорэна? Какие преимущества и недостатки имеет модель Коркорэна?

22 Какие существуют особенности модели Нельсона? Какие преимущества и недостатки имеет модель Нельсона?

23 Какие виды эмпирических моделей применяются при оценке надёжности программного обеспечения?

24 Какие существуют особенности модели фирмы IBM? Какие преимущества и недостатки имеет модель фирмы IBM?

25 Какие существуют особенности модели Холстеда? Какие преимущества и недостатки имеет модель Холстеда?

Задания для самостоятельной работы

1 В результате тестирования программы серией из 15 случайно выбранных из набора тестов обнаружено 3 ошибки. Ошибки обнаружены третьим, десятым и пятнадцатым тестами. Все ошибки исправлены сразу после обнаружения. Оцените количество оставшихся в программе ошибок, используя модель Джелинского-Моранды.

2 В результате тестирования программы серией из двенадцати случайно выбранных из набора тестов обнаружено 3 ошибки. Ошибки обнаружены вторым, пятым и двенадцатым тестами. Определите количество ошибок N в программе до начала тестирования, используя модель Джелинского-Моранды.

3 В программу преднамеренно внесено 23 ошибки. В результате тестирования обнаружено 29 ошибок, из которых 23 ошибки были внесены преднамеренно. Все обнаруженные ошибки исправлены. До начала тестирования предполагалось, что программа содержит не более 5 ошибок. Оцените количество ошибок до начала тестирования и степень отлаженности программы, используя модель Миллса.

4 В программу были преднамеренно внесено 32 ошибки. В программе перед началом тестирования было 35 ошибок. В процессе семи тестовых прогонов было выявлено следующее количество ошибок (таблица 3.1).

Таблица 3.1 – Ошибки, выявленные в процессе тестовых прогонов программы

Номер прогона	1	2	3	4	5	6	7
V	8	6	7	4	3	2	2
S	5	3	3	4	3	1	1

Оцените количество ошибок перед каждым тестовым прогоном и степень отлаженности программы после каждого прогона, используя мо-

дель Миллса. Постройте график зависимости возможного числа ошибок в программе от номера тестового прогона. Сделайте анализ динамики отлаженности программы. Постройте диаграмму и оцените тенденцию изменения отлаженности программы.

5 По результатам тестирования программы двумя независимыми группами первой группой обнаружено 23 ошибки. Количество ошибок до начала тестирования 22. Общее количество обнаруженных ошибок двумя группами 9. Определите количество ошибок, обнаруженных второй группой, используя эвристическую модель.

6 Для испытания программы использовалось 22 набора исходных данных, которые выбирались в соответствии с функцией распределения частот, представленных в таблице 3.2.

Таблица 3.2 – Исходные данные задачи

№ теста	Частота выбора теста	Исход прогона теста	№ теста	Частота выбора теста	Исход прогона теста
1	0,09	1	12	0,05	1
2	0,06	0	13	0,04	1
3	0,07	1	14	0,03	0
4	0,05	0	15	0,06	1
5	0,06	0	16	0,05	0
6	0,05	1	17	0,05	1
7	0,03	0	18	0,03	1
8	0,05	1	19	0,01	0
9	0,04	0	20	0,03	1
10	0,05	1	21	0,02	1
11	0,07	0	22	0,01	0

В двенадцати тестах обнаружены ошибки. Исходы прогонов, закончившиеся отказом, обозначены единицами. Определите надёжность программы по результатам испытаний, используя модель Нельсона.

4 Тестирование программных средств

Динамическим методом верификации программного обеспечения является тестирование программы.

4.1 Основные понятия и определения тестирования

Тестирование (testing) – процесс обнаружения ошибок в программе.

Отладка (debugging) – процесс обнаружения и исправления ошибок в программе.

Контроль (verification) – процесс обнаружения ошибки при выполнении программы в тестовой или моделируемой среде.

Испытание (validation)– процесс обнаружения ошибки при выполнении программы в заданной реальной среде.

Аттестация (certification)– авторитетное подтверждение правильности программы.

Тестирование модуля (module testing) – контроль отдельного программного модуля.

Тестирование сопряжений (integration testing) – контроль сопряжений между частями системы (модулями, компонентами, подсистемами).

Тестирование внешних функций (external function testing) – контроль внешнего поведения системы, определённого внешними спецификациями.

Комплексное тестирование (system testing) – контроль и/или испытание системы по отношению к исходным целям.

Тестовый случай (test case) – набор входных данных, на которых программа исполняется в процессе тестирования.

Тестирование приемлемости (acceptance testing) – проверка соответствия программы требованиям пользователя.

Тестирование настройки (installation testing) – процесс выявления ошибок, возникших в процессе настройки системы [6].

4.2 Этапы тестирования программ

Тестирование включает три этапа:

- создание тестового набора (test case) для среды тестирования (testing environment);
- прогон программы на тестах с получением протокола результатов тестирования (test log);
- оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.

Основная проблема тестирования — определение достаточности множества тестов и определение данных тестов [13; 14; 15; 16].

4.3 Классификация методов тестирования

Классификация методов тестирования приведена на рисунке 4.1.

Рисунок 4.1 – Классификация методов тестирования

Уровни тестирования:

- модульное тестирование (юнит-тестирование) – тестирование минимального компонента (класс, функция);
- интеграционное тестирование – тестирование интерфейса между компонентами, подсистемами или системами;
- системное тестирование – тестирование интеграционной системы на соответствие требованиям.

4.4 Структурное тестирование программ

Структурное тестирование – процесс обнаружения ошибок в логике программы.

Покрытие кода тестами – мера, показывающая, на сколько процентов исходный код программы был протестирован.

Стратегия «белого ящика», или стратегия тестирования, управляемого логикой (текста) программы, позволяет исследовать внутреннюю структуру программы. Тестирование заключается в проверке каждого пути, каждой ветви алгоритма. Внешняя спецификация во внимание не принимается. Тестирующий получает тестовые данные посредством анализа программы [6].

4.4.1 Критерии структурного тестирования

Структурные критерии используют модель программы в виде «белого ящика». Структурные критерии базируются на основных элементах управляющего потокового графа, операторах, ветвях и путях:

- критерий тестирования команд (критерий C0) – набор тестов, обеспечивающий прохождение команды не менее одного раза;
- критерий тестирования ветвей (критерий C1) – набор тестов, обеспечивающий прохождение каждой ветви не менее одного раза.
- критерий тестирования путей (критерий C2) – набор тестов, обеспечивающий прохождение каждого пути не менее одного раза [16-20].

4.4.2 Методы структурного тестирования

Методы структурного тестирования:

- покрытие операторов;
- покрытие решений;
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.

Метод покрытия операторов выполняет каждый оператор хотя бы один раз. Покрытие операторов – слабый критерий, так как выполнение оператора хотя бы один раз есть необходимое, но недостаточное условие для тестирования по стратегии «белого ящика».

Метод покрытия решений выполняет каждое направление перехода хотя бы один раз. Покрытие решений (покрытие переходов) – сильный критерий покрытия логики программы. Покрытие решений удовлетворяет критерию покрытия операторов. При выполнении каждого направления перехода оператор должен быть выполнен.

Исключения составляют:

- программа не имеет решений;
- программы с несколькими точками входа;
- операторы переключатели switch.

Покрытие решений требует, чтобы каждое решение имело результатом значение истина или ложь, и каждый оператор выполнялся по крайней мере один раз.

Метод покрытия условий выполняет каждое условие хотя бы один раз. Покрытие условий – сильный критерий покрытия логики программы. Покрытие условий удовлетворяет критерию покрытия решений. Покрытие условий предусматривает число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз. Каждой точке входа в программу, а также switch-операторам должно быть передано управление при вызове, по крайней мере один раз.

Метод покрытия решений/условий выполняет условия и результаты решения по крайней мере один раз, и каждой точке входа передаётся управление по крайней мере один раз. Покрытие решений/условий требует набора тестов, чтобы все возможные результаты каждого условия в решении, все результаты каждого решения выполнялись, по крайней мере, один раз, и каждой точке входа передавалось управление, по крайней мере, один раз. Покрытие решений/условий невозможно применить для выполнения всех результатов всех условий вследствие того, что определённые условия скрыты другими условиями.

Метод комбинаторного покрытия условий выполняет покрытия решений, покрытия условий и покрытия решений/условий. Комбинаторное покрытие условий требует создания набора тестов, чтобы все возможные комбинации результатов условия в каждом решении, и все точки входа выполнялись, по крайней мере, один раз [17].

4.5 Функциональное тестирование программ

Функциональное тестирование – процесс обнаружения ошибок в функциях программы на всей области определения.

Тесты проектируются на основе внешних спецификаций программ и модулей, или спецификаций сопряжения модуля с другими модулями, программа рассматривается как чёрный ящик. Тест проверяет соответствие программы внешним спецификациям. Содержание программного модуля не имеет значения.

4.5.1 Функциональные критерии

Функциональный критерий обеспечивает контроль степени выполнения требований заказчика в программном продукте. Функциональные критерии подразделяются на виды:

- тестирование проектов спецификации – набор тестов, обеспечивающий проверку каждого тестируемого пункта не менее одного раза;
- тестирование классов входных данных – набор тестов, обеспечивающий проверку представителя каждого класса входных данных не менее одного раза;
- тестирование правил – набор тестов, обеспечивающий проверку каждого правила, если входные и выходные значения описываются набором правил грамматики;
- тестирование классов выходных данных – набор тестов, обеспечивающий проверку представителя каждого выходного класса, при условии, что выходные результаты заранее расклассифицированы, отдельные классы результатов учитывают ограничения на ресурсы или на время (time out);
- тестирование функций – набор тестов, обеспечивающий проверку каждого действия, реализуемого тестируемым модулем, не менее одного раза;
- комбинированные критерии для программ и спецификаций – набор тестов, обеспечивающий проверку всех комбинаций непротиворечивых условий программ и спецификаций не менее одного раза [16; 21; 22; 23].

4.5.2 Методы функционального тестирования

Методы функционального тестирования:

- эквивалентное разбиение (классы эквивалентности);
- анализ граничных значений;
- метод функциональных диаграмм;

- предположение об ошибке.

Эквивалентное разбиение – метод функционального тестирования программы, основанный на разбиении области определения исходных данных функции на классы эквивалентности.

Класс эквивалентности – набор данных с общими свойствами.

Положения метода эквивалентного разбиения классов:

- исходные данные разбиваются на конечное число классов эквивалентности. В одном классе эквивалентности содержатся такие тесты, что если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку;
- каждый тест должен включать максимальное количество классов эквивалентности, чтобы минимизировать общее число тестов.

Разработка тестов методом эквивалентного разбиения классов производится двумя этапами:

- выделение классов эквивалентности;
- построение теста.

Классы эквивалентности выделяются посредством выбора каждого входного условия, которые выбираются из технического задания или спецификации и разбиваются на две и более группы (таблица 4.1).

Таблица 4.1 – Классы эквивалентности

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности

Определение классов эквивалентности осуществляется эвристическим способом.

Правила определения классов эквивалентности:

- если входное условие задаёт диапазон значений, то определяют один правильный класс эквивалентности и два неправильных;
- если входное условие задаёт конкретное значение, то определяют один правильный и два неправильных класса эквивалентности;
- если входное условие задаёт множество значений, то определяют один правильный и один неправильный классы эквивалентности;
- если входное условие задаёт булево значение, то определяют один правильный и один неправильный классы эквивалентности.

После определения классов эквивалентности разрабатываются тестовые варианты. Тестовый вариант выбирается таким образом, чтобы проверить наибольшее количество свойств классов эквивалентности.

Определение тестовых вариантов:

- каждому классу эквивалентности присваивается уникальный номер;
- если еще остались не включенные в тесты правильные классы, то пишутся тесты, которые покрывают максимально возможное количество классов;
- если остались не включенные в тесты неправильные классы, то пишут тесты, которые покрывают только один класс.

Анализ граничных значений – метод функционального тестирования программного обеспечения, основанный на создании тестовых вариантов, анализирующих граничные значения. Обнаружение ошибок производится на границах области ввода данных. Анализ граничных значений дополняет метод классов эквивалентности.

Отличительные особенности анализа граничных значений от эквивалентного разбиения:

- тестовые варианты создаются для проверки рёбер классов эквивалентности;
- при создании тестовых вариантов учитывают не только условия ввода, но и области вывода.

Метод требует определённой степени творчества и специализации в рассматриваемой задаче.

Правила анализа граничных значений:

- если условие ввода задаёт диапазон значений, то тестовые варианты определяются для значений чуть левее нижней границы диапазона и значения чуть правее верхней границы диапазона;
- если условие ввода задаёт дискретное множество значений, то создаются тестовые варианты для проверки минимального и максимального значений и для значений чуть меньше минимума и чуть больше максимума;
- для условий вывода применяются правила 1 и 2. Например, если необходимо вывести таблицу значений, где количество строк и столбцов меняется, то задаётся тестовый вариант для минимального вывода и тестовый вариант для максимального вывода;
- если внутренние структуры данных программы имеют предписанные границы, то разрабатываются тестовые варианты, проверяющие структуры на границах.
- если входные или выходные данные программы являются упорядоченными множествами, то тестируют первый и последний элементы множеств.

Преимущество анализа граничных значений – обнаружение большого числа ошибок.

Недостатки анализа граничных значений:

- определение границ для задачи является трудоёмким процессом;
- отсутствие проверки комбинации входных значений.

Метод функциональных диаграмм – метод функционального тестирования программного обеспечения, основанный на проектировании тестовых вариантов, использующих формальную запись логических условий и соответствующих действий.

Этапы метода функциональных диаграмм:

1 для каждого модуля перечисляются причины (условия ввода, классы эквивалентности условий ввода) и следствия (действия или условия вывода). Идентификатор присваивается причине и следствию;

2 разрабатывается диаграмма причинно-следственных связей;

3 определяется таблица решений на основе диаграммы причинно-следственных связей;

4 столбцы таблицы решений преобразуются в тестовые варианты.

Таблица решений снабжается примечаниями, задающими ограничения и описывающими комбинации, которые невозможны.

Преимущество метода функциональных диаграмм – наглядность диаграммы причинно-следственных связей.

Недостаток метода функциональных диаграмм – плохое исследование граничных условий.

Предположение об ошибке – метод функционального тестирования программного обеспечения, основанный на опыте и интуиции эксперта-профессионала. Тестировщик с большим опытом и стажем тестирования программ обнаруживает ошибки, подсознательно используя метод предположения об ошибке. Сущность метода заключается в том, чтобы составить список, перечисляющий возможные ошибки и ситуации, в которых ошибки могли проявиться. На основе списка разрабатываются тестовые варианты [17].

4.6 Интеграционное тестирование программ

Интеграционное тестирование – тестирование программного обеспечения на этапе сборки модулей в единый комплекс. Цель интеграционного тестирования – нахождение ошибок взаимодействия модулей (компонент, подсистем). Тестирование выполняется через интерфейс модулей с использованием метода «чёрного ящика» [19].

4.6.1 Методы интеграционного тестирования

Существуют два метода сборки модулей:

- монолитный, характеризующийся одновременным объединением модулей в тестируемый комплекс;
- инкрементальный, характеризующийся пошаговым (помодульным) наращиванием комплекса программ с пошаговым тестированием собираемого комплекса [16].

В инкрементальном методе выделяют две стратегии добавления модулей:

- восходящее тестирование;
- нисходящее тестирование.

При восходящем тестировании сначала тестируются терминальные модули, не зависящие от других модулей, затем тестируются модули, которые зависят от проверенных модулей. Для модуля разрабатывают драйвер.

Драйвер – программный модуль, эмулирующий окружение модуля при тестировании. Драйвер вызывает тестируемый модуль, передает тестовые данные, проверяет результаты работы модуля и корректность реализации его интерфейса. При восходящем тестировании упрощается локализация ошибок.

При нисходящем тестировании сначала тестируют верхний управляющий модуль программной системы без модулей низкого уровня, вместо которых используют заглушки, затем тестируют модули более низкого уровня.

Заглушка – программный модуль, обладающий тем же интерфейсом, что и замещаемый модуль нижележащего уровня. Заглушка не реализует функциональность замещаемого модуля, а возвращает значения, позволяющие проверить функционирование тестируемого модуля [19].

Контрольные вопросы

1 Что называется тестированием программного обеспечения?

2 Что называется отладкой программного обеспечения?

3 В чём заключается отличие тестирования от отладки?

4 Что понимается под испытанием?

5 Что называется тестовым случаем?

6 Какие этапы включает процесс тестирования программного обеспечения?

7 Каким образом классифицируются методы тестирования?

8 Что называется структурным тестированием?

9 Какие критерии применяются при структурном тестировании?

10 Какие методы используются при структурном тестировании?

11 Что называется функциональным тестированием?

12 Какие критерии применяются при функциональном тестировании?

13 Какие методы используются при функциональном тестировании?

14 Что называется интеграционным тестированием?

15 Какие методы используются при интеграционном тестировании?

Задания для самостоятельной работы

1 Разработайте программу на языке C++ расчёта значения функции, заданной графическим способом (рисунок 4.2). Проведите структурное тестирование программного приложения.

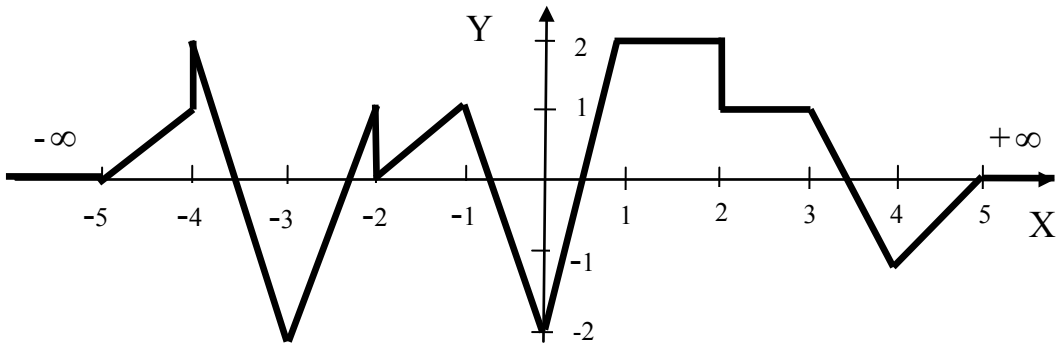


Рисунок 4.2 – Графический способ задания функций

2 Разработайте программу на языке C++ определения принадлежности точки с введенными координатами закрашенной области (рисунок 4.3). Проведите структурное тестирование программного приложения.

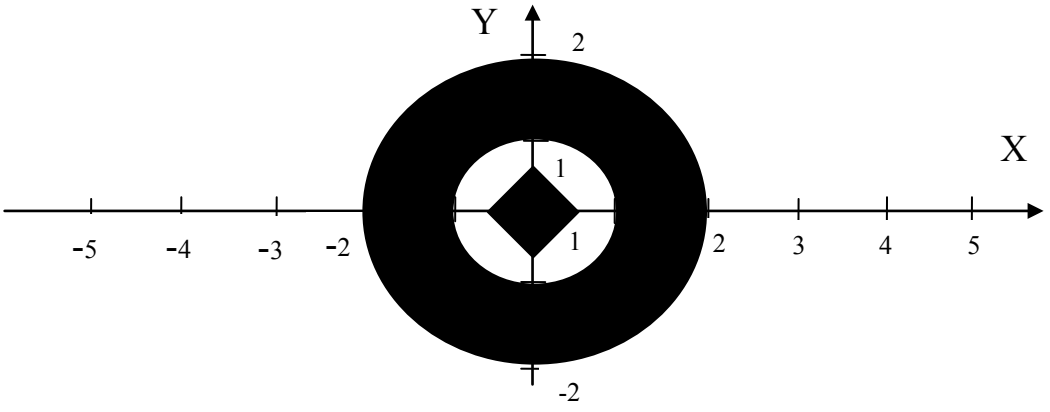


Рисунок 4.3 – Графическое представление закрашенной области

3 Разработайте программу на языке C++ решения алгебраического уравнения 2-й степени (квадратное уравнение) $a * x^2 + b * x + c = 0$. Проведите функциональное тестирование программы.

4 Разработайте программу на языке С++ решения алгебраического уравнения 3-й степени (кубическое уравнение) $a * x^3 + b * x^2 + c * x + d = 0$. Корни приведенного уравнения рассчитать по формулам Кардано. Проведите функциональное тестирование программы.

5 Разработайте программу на языке С++ решения алгебраического уравнения 3-й степени (кубическое уравнение) $a * x^3 + b * x^2 + c * x + d = 0$. Корни рассчитать по тригонометрической формуле Виета. Проведите функциональное тестирование программы.

6 Разработайте программу на языке С++ решения биквадратного уравнения $a * x^4 + b * x^2 + c = 0$. Проведите функциональное тестирование программы.

7 Разработайте программу на языке С++. В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

1) сумму положительных элементов массива;

2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию. Проведите интеграционное тестирование программного приложения.

8 Разработайте программу на языке С++. Дана целочисленная прямоугольная матрица $[A_{n*m}]$. Определить:

1) количество элементов матрицы, содержащих значения больше С;

2) максимум элементов главной и побочной диагоналей матрицы;

3) суммы элементов сторон и диагоналей элементов матрицы;

Суммы элементов сторон и диагоналей матрицы упорядочите по возрастанию простым методом обмена. Проведите интеграционное тестирование программного приложения.

5 Стандартизация качества программного обеспечения

5.1 Основные понятия и определения

Стандартизация – деятельность, направленная на разработку и установление требований, норм, характеристик, как обязательных для выполнения, так и рекомендуемых, обеспечивающая право потребителя на приобретение товаров надлежащего качества, а также право на безопасность и комфортность труда.

Цель стандартизации – достижение оптимальной степени упорядочения в той или иной области посредством широкого и многократного использования установленных положений, требований, норм для решения реально существующих, планируемых или потенциальных задач.

Основные результаты деятельности по стандартизации – повышение степени соответствия продукта (услуги), процессов их функциональному назначению, устранение технических барьеров в международном товарообмене, содействие научно-техническому прогрессу и сотрудничеству в различных областях.

Объект стандартизации – продукция, процесс, услуга, для которых разрабатывают требования, характеристики, параметры, правила.

Область стандартизации – совокупность взаимосвязанных объектов стандартизации.

Стандарт – нормативно-технический документ, устанавливающий комплекс норм, правил и требований к объекту стандартизации, разработанный на основе согласия и на обобщенных результатах научных исследований, технических достижений и практического опыта, направленный на достижение оптимальной пользы для общества.

Уровень стандартизации – вид стандартизации, зависящий от географического, экономического, политического региона участников, принимающих участие в стандартизации.

Схема уровней стандартизации приведена на рисунке 5.1.

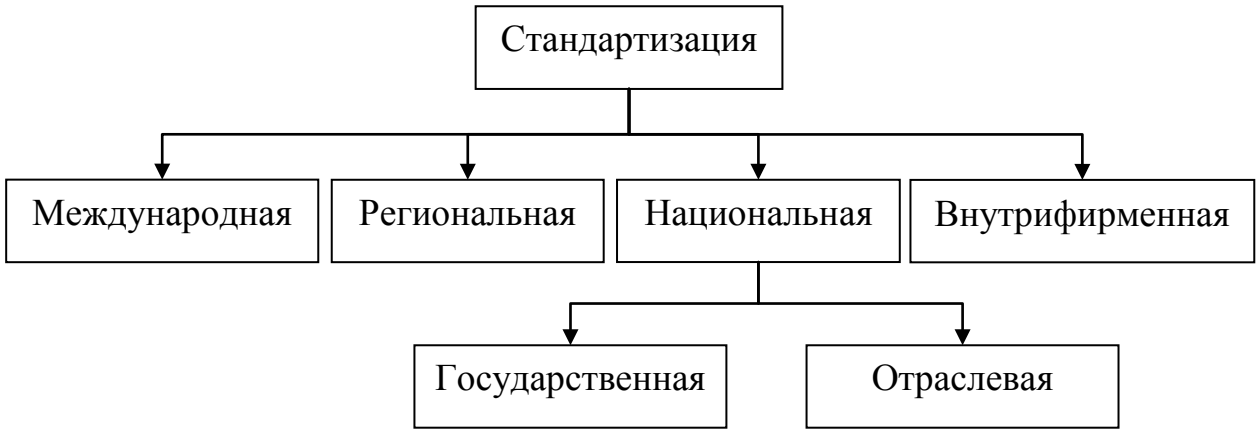


Рисунок 5.1 – Схема уровней стандартизации

Международная стандартизация – стандартизация, открытая для соответствующих органов любой страны.

Региональная стандартизация – стандартизация, открытая для соответствующих органов государств одного географического, политического или экономического региона.

Национальная стандартизация – стандартизация в одном государстве. Национальная стандартизация подразделяется на виды:

- государственная;
- отраслевая.

Внутрифирменная стандартизация – стандартизация внутри фирмы.

Внутрифирменные стандарты – набор внутрифирменных инструкций и руководств значительного объема, которые постоянно корректируются в целях совершенствования и изменения среды их применения [6].

5.2 Роль стандартизации в управлении качеством

При разработке программных средств для достижения заданных требований необходимо планирование и управление обеспечением качества в течение всего цикла создания программ.

Для управления качеством требуется классифицировать критерии в зависимости от классов программ и связать их с методами и этапами разработки. Набор показателей качества программных средств зависит от функционального назначения и свойств каждого программного средства. Критерий применяется, если определена метрика и указан способ измерения. Основной метод измерения качества программ является тестирование. На рисунке 5.2 приведена схема процесса управления качеством программных средств.

Качество программных средств обеспечивается стандартами:

- международные (ISO, IEC, ECMA);
- национальные (ГОСТ, DOD, MIL, ANSI, IEEE).

Группы стандартов, регламентирующих качество программ:

- общие для любых программ;
- формализующие показатели качества программ;
- отражающие планирование, методы и технологию управления качеством программ;
- поддерживающие технологический процесс создания сложных программных средств высокого качества.

Общие стандарты – стандарты, обеспечивающие качество любых программ независимо от их свойств, назначения и характеристик. Стандарты качества программных средств как объектов разработки и производства ориентируются в терминологии, структуре и содержании на общие стандарты. Общие стандарты – международные стандарты «ISO 8402. Управление качеством и обеспечение качества – Словарь», «ISO 9000. Системы менеджмента качества. Основные положения и словарь», «ISO 9001. Системы менеджмента качества. Требования».

Рисунок 5.2 – Схема процесса управления качеством программных средств

Стандарты, формализующие показатели качества программ, – стандарты, формализующие номенклатуру, понятия и содержание показателей качества различных классов программ. Показатели качества строятся иерархически по уровням: факторы, критерии, метрики. Показатели качества программ ориентируются на сложные программные средства высокого качества и на выбор из них разработчиком конкретного набора в зави-

симости от требований заказчика. Международные стандарты «ISO/IEC 9126. Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению», «ANSI/IEEE 729. Глоссарий стандартизированных терминов по технике разработки программного обеспечения», «ANSI/IEEE 1061. Система показателей качества программного обеспечения.», «ANSI/IEEE1044. Стандартная классификация программных ошибок, отказов и сбоев» и отечественный стандарт «ГОСТ 28195-89. Оценка качества программных средств. Общие положения» – стандарты, формализующие показатели качества программ.

Стандарты, отражающие планирование, методы и технологию управления качеством программ, – стандарты, регламентирующие методы, технологию и документацию для планирования и управления обеспечением качества программных средств. Международные стандарты «ISO/IEC 688. Управление передачей информации между фазами жизненного цикла программных средств», «ANSI/IEEE 1058 Планы управления проектами создания программного обеспечения», «DI-S-30567A. План разработки программ для ЭВМ», «ISO/IEC 687. Управление конфигурацией программного обеспечения», «ANSI/IEEE 828. План управления конфигурацией программного обеспечения».

Стандарты, поддерживающие технологический процесс создания сложных программных средств высокого качества, – стандарты, поддерживающие процесс создания сложных программных средств и отдельные этапы. Стандарты регламентируют тестирование и сертификацию программных компонент и программных средств, и документирование процессов [24].

В международном стандарте ISO 9126:1991 при выборе показателей качества учитываются принципы:

- ясность и измеряемость значений;
- отсутствие перекрытия между показателями;
- соответствие понятиям и терминологии;
- возможность уточнения и детализации.

Шесть характеристик оценивают программные средства с позиции пользователя, разработчика и управляющего проектом. Характеристики подразделяются на 21 субхарактеристику программных средств.

В стандарте ГОСТ 28195-89 на первом уровне выделены 6 показателей качества: надёжность, корректность, удобство применения, эффективность, универсальность, сопровождаемость. Показатели качества подразделяются на 19 критериев качества второго уровня. Критерии качества детализируются

240 метриками и оценочными элементами, принимающими значения от 0 до 1. Показатели, критерии и метрики выбираются в зависимости от назначения, функций и этапов жизненного цикла программных средств.

В стандарте ГОСТ 28806-90 формализуются общие понятия программы, программного средства, программного продукта и качества. Приводятся определения 18 наиболее употребляемых терминов, связанных с оценкой характеристик программ. Уточнены понятия базовых показателей качества, приведённых в ГОСТ 28195-89 [6].

Международные стандарты качества, безопасности, тестирования, процессов жизненного цикла программных средств, документации при тестировании программ приведены в таблице 5.1 [24].

Таблица 5.1 – Международные стандарты

Стандарт	Описание
ISO 09126:1991	Оценка программного продукта. Характеристики качества и руководство по их применению
DOD-STD-2168	Программа обеспечения качества оборонных программных средств
ISO 09000-:1991	Общее руководство качеством и стандарты по обеспечению качества. Ч.3: Руководящие указания по применению ISO 09001 при разработке, поставке и обслуживанию программного обеспечения
ISO 12207:1995	Процессы жизненного цикла программных средств
DOD-STD 2167A:1988	Разработка программных средств для систем военного назначения
ISO 09646-1-6:1991.ИТ.ВОС	Методология и основы аттестационного тестирования ВОС
ANSI/IEEE 829-83	Документация при тестировании программ
ANSI/IEEE 1008-86	Тестирование программных модулей и компонент программных средств
ANSI/IEEE 1012-86	Планирование проверки (оценки) (verification) и подтверждения достоверности (validation) программных средств

5.3 Оценка качества программного обеспечения в соответствии с требованиями стандарта

Методика оценки качества программного обеспечения основывается на требованиях стандарта ГОСТ 28195-89.

Процесс оценки качества программного обеспечения производится для каждой стадии жизненного цикла и включает:

- выбор показателей качества;
- определение значений показателей;
- сравнение полученных значений с базовыми значениями показателей качества.

В соответствии с ГОСТ 28195-89 период жизненного цикла программного обеспечения подразделяется на временные промежутки (фазы):

1 Анализ – этап определения требований к программному обеспечению, спецификация требований и формирования технического задания на проектирование программы.

2 Проектирование – этап разработки технического проекта.

3 Реализация – этап разработки программного обеспечения, средств тестирования и документации.

4 Тестирование – этап испытания программы и устранения недостатков.

5 Изготовление – этап преобразования программного обеспечения в форму, готовую для поставки, завершение формирования документации.

6 Внедрение – этап подтверждения стабильной работы программного обеспечения и ввод в стадию использования.

7 Эксплуатация – этап применения программного обеспечения по назначению.

8 Сопровождение – этап устранения дефектов в процессе эксплуатации, усовершенствование и модификация программного обеспечения

Оценка качества программного обеспечения на стадиях жизненного цикла осуществляется на основе четырёхуровневой системы показателей. Схема показателей качества в соответствии с ГОСТ 28195-89 приведена на рисунке 5.3.

Показатели первого уровня (факторы качества) характеризуют потребительски-ориентированные свойства программных средств, которые соответствуют потребностям пользователей. Факторы качества определяют значимые свойства программы. Фактор представляет собой интегральную оценку, которой соответствует несколько критериев качества (комплексных показателей второго уровня) [12].

Рисунок 5.3 – Схема показателей качества программного обеспечения

Порядок расчёта числовых значений производится следующим образом:

1 Определить критерии фаз жизненного цикла программы в соответствии с ГОСТ 28195-89.

2 Определить набор метрик критериев в соответствии с ГОСТ 28195-89.

3 Определить набор оценочных элементов метрик в соответствии с ГОСТ 28195-89..

4 Определить численные значения оценочных элементов, используя рекомендации ГОСТ 28195-89.

5 Рассчитать итоговые значения метрик M_k на основе исходных значений оценочных элементов по формуле 5.5.

6 Рассчитать абсолютные значения критериев P_j на основе вычисленных значений метрик и заданных величин весовых коэффициентов метрик V_k^M по формуле 5.4.

7 Рассчитать относительное значение критерия фактора K_j по формуле 5.3.

8 Рассчитать численное значение фактора R^{ϕ} по заданным значениям весовых коэффициентов V_j^k по формуле 5.2.

9 Сделать выводы о качестве программного обеспечения на основе расчётных данных.

Последовательность расчётов числовых значений представляется в виде

$$m_t \rightarrow m_i \rightarrow M_k \rightarrow P_j \rightarrow K_j \rightarrow R^{\phi}, \qquad (5.1)$$

где m_t – отдельные значения оценочного элемента, выставленные t-м экспертом;

m_i – значения оценочных элементов, проставленных группой экспертов;

Mk – итоговые значения метрики, определяемые на основе оценочных элементов;

P_j – абсолютное значение критериев, определяемое на основе значений соответствующих метрик;

K_j – относительные значения критерия качества;

R^{ϕ} – численное значение фактора, определяемое на основе значений критериев.

Численное значение фактора, определяемое на основе значений критериев, определяется по формуле

$$R^{\phi} = \sum_{j=1}^{N} (K_j * V_j^k), \qquad (5.2)$$

где K_j – относительное значение j-го критерия качества;

V_j^k – весовой коэффициент j-го критерия качества, $j = \overline{1, N}$;

N – количество критериев, входящих в состав фактора.

Ограничения, наложенные на весовой коэффициент критерия качества:

- сумма весовых коэффициентов j-го критерия качества равна 1; $\sum_{j=1}^{N} V_j^k = 1$.
- весовой коэффициент j-го критерия качества V_j^k подбирается на основе субъективной оценки значимости каждого критерия качества.

Критерии – характеристики программных средств, обеспечивающие достижение свойств.

Относительное значение критерия фактора определяется по формуле

$$K_j = \frac{P_j}{P_j^{баз}}, \qquad (5.3)$$

где P_j – абсолютное значение j-го критерия качества для программного средства в соответствии с рассчитываемым фактором качества;

$P_j^{баз}$ – базовое (эталонное) j-го критерия качества, с которым сравнивается критерий оцениваемого программного средства, $j = \overline{1, N}$;

Абсолютное значение критериев P_j определяется на основе значений соответствующих метрик по формуле

$$P_j = \sum_{k=1}^{n} (M_k * V_k^M), \qquad (5.4)$$

где M_k – итоговое значение метрик соответствующего критерия, $k = \overline{1, n}$;

V_k^M – весовые коэффициенты уровня метрик качества;

n – количество метрик, входящих в состав критерия.

Ограничения, наложенные на весовой коэффициент уровня метрик качества:

- сумма весовых коэффициентов уровня метрик качества равна 1; $\sum_{k=1}^{n} V_k^M = 1$.
- величина V_k^M определяется на основе субъективной оценки значимости метрик качества, входящей в состав оцениваемого критерия.

Метрики – показатели качества, представляющие абсолютную меру количественной оценки заданного критерия.

Итоговые значения метрики M_k определяются на основе оценочных элементов по формуле

$$M_k = \frac{\sum_{i=1}^{Q} m_i^k}{Q}, \tag{5.5}$$

где m_i^k – среднее значение i-го оценочного элемента для k-й метрики, $i = \overline{1, Q}$;

Q – количество оценочных элементов k-й метрики.

Величина оценочного элемента рассчитывается по формуле

$$m_i = \frac{\sum_{t=1}^{N} m_t}{N}, \tag{5.6}$$

где m_t – отдельные значения оценочного элемента, выставленные t-м экспертом, $t = \overline{1, N}$;

N – количество экспертов [25].

Контрольные вопросы

1 Что называется стандартизацией?

2 В чём заключается цель стандартизации?

3 Какие существуют виды стандартизации?

4 Какие группы стандартов обеспечивают качество программных средств?

5 Какая разработана схема процесса управления качеством программных средств?

6 Какие принципы учитываются при выборе показателей качества в соответствии с международным стандартом ISO 9126-1991?

7 Какие показатели качества программных средств представлены в стандарте ГОСТ 28195-89?

8 Какие этапы включает процесс оценки качества программного обеспечения для каждой стадии жизненного цикла?

9 Какие фазы включает жизненный цикл программного обеспечения?

10 Какие шаги включает порядок расчёта числовых значений оценок качества программного обеспечения на основе четырёхуровневой системы показателей качества?

Задания для самостоятельной работы

Для выполнения заданий рекомендуется использовать приложения А и Б учебного пособия или стандарт ГОСТ 28195-89.

1 Провести оценку качества программы на основе фактора «корректность фазы сопровождения (обслуживание)». Базовые показатели корректности по критерию «полнота реализации» равны 0,7, по критерию «согласованность» – 0,6, по критерию «проверенность» – 0,8. Метрики для одного и того же критерия и все критерии корректности имеют одинаковые коэффициенты важности, сумма значений которых на каждом уровне равна единице. Количество экспертов в группе – пять человек.

2 Провести оценку качества программы на основе факторов «удобство применения» и «эффективность фазы изготовления». Базовые показатели удобства применения по критерию «лёгкость освоения» равны 0,85, по критерию «доступность эксплуатационных документов» – 0,68, по критерию «удобство эксплуатации и обслуживания» – 0,58. Базовые показатели эффективности по критерию «уровень автоматизации» равен 0,53, по критерию «временная эффективность» – 0,91, по критерию «ресурсоёмкость» – 0,82. Метрики для одного и того же критерия и все критерии имеют коэффициенты важности, вводимые пользователем. Сумма значений коэффициентов важности на каждом уровне равна единице. Количество экспертов в группе – пять человек.

3 Провести оценку качества программы на основе факторов «универсальность» и «корректность фазы тестирования». Базовые показатели фактора «универсальность» по критериям «гибкость», «мобильность»,

«модифицируемость» и фактора «корректность» по критериям «полнота реализации», «согласованность» и «проверенность фазы тестирования» вводятся пользователем. Метрики для одного и того же критерия и все критерии имеют коэффициенты важности, вводимые пользователем. Сумма значений коэффициентов важности на каждом уровне равна единице. Количество экспертов в группе – пять человек.

4 Провести оценку качества программы на основе факторов «эффективность» и «корректность фазы анализа». Базовые показатели фактора «эффективность» по критериям «уровень автоматизации», «временная эффективность», «ресурсоёмкость» и фактора «корректность» по критериям «полнота реализации», «согласованность» и «проверенность фазы анализа» вводятся пользователем. Метрики для одного и того же критерия и все критерии имеют коэффициенты важности, вводимые пользователем. Сумма значений коэффициентов важности на каждом уровне равна единице. Количество экспертов в группе – пять человек.

5 Провести оценку качества программы на основе факторов «сопровождаемость», «удобство применения» и «универсальность» фазы «реализация». Базовые показатели факторов по критериям фазы «реализация» вводятся пользователем. Метрики для одного и того же критерия и все критерии имеют коэффициенты важности, вводимые пользователем. Сумма значений коэффициентов важности на каждом уровне равна единице. Количество экспертов в группе – шесть человек.

6 Сертификация программного обеспечения

6.1 Основные понятия, цели и виды сертификации программных средств

Основные понятия и определения приведены в стандарте ISO/IEC 00002 – Общие термины и определения в области стандартизации и смежных видов деятельности.

Сертификация соответствия – действие третьей стороны, доказывающее, что обеспечивается необходимая уверенность в том, что должным образом идентифицированная продукция, процесс или услуга соответствует конкретному стандарту или нормативному документу.

Сертификат соответствия – документ, изданный в соответствии с правилами системы сертификации, удостоверяющий, что обеспечивается необходимая уверенность в том, что должным образом идентифицированная продукция, процесс или услуга соответствует конкретным стандартам или нормативным документам.

Цель сертификации технологий проектирования и производства систем и программных средств – защита интересов пользователей, государ-

ственных и ведомственных интересов на основе контроля качества продуктов, обеспечения высоких потребительских свойств, повышения эффективности затрат в сфере их производства, эксплуатации и сопровождения, повышения объективности оценок характеристик и обеспечения конкурентоспособности конечного продукта.

При анализе и организации процессов сертификационных испытаний и/или объектов системы и комплекса программ следует учитывать базовые компоненты методологии сертификации:

- цели сертификации – правовые, экономические, формальные;
- проблемы, которые необходимо решать для обеспечения высокой эффективности и достоверности результатов сертификационных испытаний;
- исходные данные и документы, необходимые для проведения сертификации: стандарты и нормативные документы, их структура и содержание;
- характеристики и классификацию продуктов и/или процессов сертификации и их показатели качества;
- ресурсы обеспечения и испытаний – финансовые, кадры специалистов, их аппаратурная оснащённость, нормативные и инструментальные средства.

Виды сертификации:

- обязательная сертификация;
- добровольная сертификация.

Обязательная сертификация необходима для программных продуктов и их производства, выполняющих ответственные функции, в которых недостаточное качество, ошибки или отказы могут нанести большой ущерб или опасны для жизни и здоровья людей. Обязательная сертификация программных продуктов способствует значительному снижению риска заказчика и повышению безопасности функционирования программного продукта у потребителя до необходимого уровня.

Добровольная сертификация применяется с целью повышения конкурентоспособности продукции, расширения сферы её использования и получения экономических преимуществ. Экономическими целями сертификации являются большие тиражи изделий при производстве, большая длительность жизненного цикла с множеством версий, снижение налогов за высокое качество, увеличение прибыли разработчиков и поставщиков программного продукта, сокращение рекламаций пользователей [26].

6.2 Требования к качеству функционирования программных продуктов

Сертификация программного продукта должна обеспечивать высокое качество результатов производства и полное соответствие требованиям заказчика и пользователей.

Сертификационные испытания включают этапы:

- формирование функциональных и конструктивных требований к программным продуктам, требования к допустимым рискам и проверку корректности;
- организация сертификационных испытаний программных продуктов на соответствие требованиям, предварительный выбор стратегии, планирование и оценки затрат на испытания;
- подготовка сертификационных испытаний программных продуктов, выбор методов, требования к тестам и процессы генерации динамических тестов внешней среды в реальном времени;
- программы, методики и процессы испытаний программных продуктов, а также эксплуатационной документации на соответствие требованиям к программным продуктам;
- завершение испытаний, оформление отчётной документации и утверждение акта результатов сертификационных испытаний программных продуктов на соответствие требованиям, тиражирование и внедрение сертификационных версий программных продуктов [26].

Для сертификации программных продуктов необходимо определить свойства, выделить и формализовать характеристики. Основные требования к качеству комплексов программ для сертификации определяют базовые международные стандарты (приложение А) [26].

6.3 Методики оценки качества программных средств при сертификации

В качестве объекта оценки качества работы программно-технических средств используется информационная система. Методики оценки качества программно-технических средств приведены на рисунке 6.1.

Виды проверки программного обеспечения:

- проверка на отсутствие опасных закладных элементов;
- проверка на соответствие функциональных возможностей декларируемым в документации.

Оценка качества функционирования включает:

- надёжность представления выходной информации;

- своевременность представления выходной информации;
- полноту отражения в базе данных объектов учёта предметной области;
- безошибочность входной информации;
- защищённость ресурсов от программных вирусов;
- актуальность базы данных;
- защищённость ресурсов от несанкционированного доступа [12].

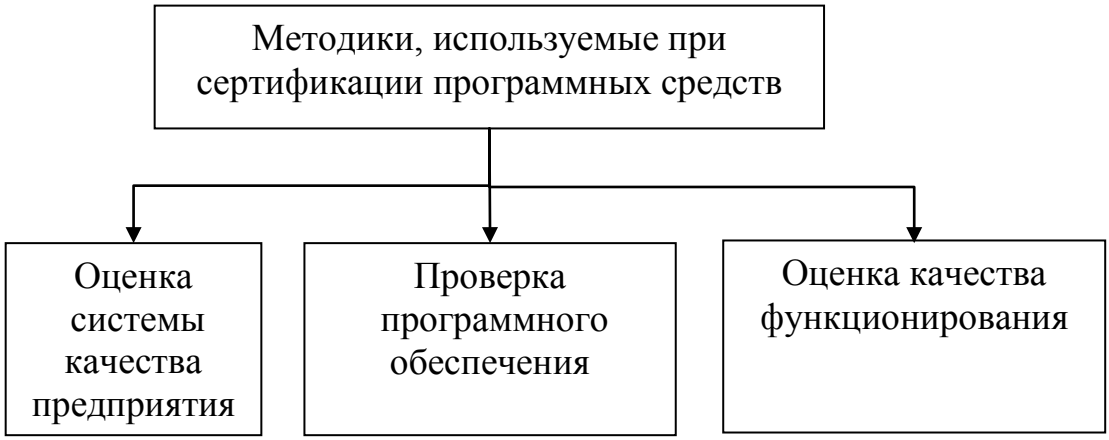


Рисунок 6.1 – Методики оценки качества программно-технических средств информационных систем

Разработанные методики требуют большого перечня входной информации для оценки качества программно-технических средств. Подготовка большого объёма входных данных затрудняет процесс оценки качества программно-технических средств [12].

6.4 Критерии оценки качества

Для упрощения процесса оценки качества программно-технических средств выбирается значимый критерий – своевременность представления запрашиваемых выходных данных информационных систем.

Метод определяет вероятностно-временные характеристики функционирования программно-технических средств. Получение данных производится по запросам с фиксированным временем выполнения. Моменты поступления запросов носят случайный характер.

Для оценки качества информационных систем применяются два критерия:

- своевременное представление информации, если среднее время T_{ij} обработки запроса на получение выходного документа i-го типа j-го приоритета не более заданного. В этом случае выполняется условие

$$T_{ij} \le t_i^{зад}, \qquad (6.1)$$

где T_{ij} – среднее время T_{ij} обработки запроса на получение выходного документа i-го типа j-го приоритета;

$t_i^{зад}$ – заданное время.

- своевременнное представление информации, если вероятность представления выходного документа i-го типа j-го приоритета за заданное время окажется не менее установленной. В этом случае выполняется условие

$$P(T_{ij} \le t_i^{зад}) \ge P_i^{треб}, \qquad (6.2)$$

где $P(T_{ij} \le t_i^{зад})$ – вероятность представления выходного документа i-го типа j-го приоритета за заданное время;

$P_i^{треб}$ – заданная вероятность [12].

6.5 Модели обслуживания запросов информации

Для оценки своевременности представления информации по критерию (6.1) используются модели:

- модель бесприоритетного обслуживания;
- модель обслуживания с относительными приоритетами;
- модель обслуживания с абсолютными приоритетами.

6.5.1 Модель бесприоритетного обслуживания запросов информации

Модель бесприоритетного обслуживания запросов информации – модель обработки запросов выходных данных без приоритетов. При поступлении нового запроса не происходит прерывания обработки предыдущего запроса. Запросы обслуживаются в порядке поступления в информационную систему.

При решении задачи определяется показатель качества – среднее время обработки запроса. Поступление запросов последовательное, по од-

ному запросу каждого типа. Обслуживание запросов осуществляется в порядке появления в очереди.

Алгоритм решения задачи бесприоритетного обслуживания запросов выходных данных информационной системы:

1 Построение диаграммы поступления и обслуживания запросов выходной информации.

2 Определение T_i моментов времени окончания обслуживания запросов выходной информации. Переменная i – номер запроса выходной информации, $i = \overline{1, n}$. Величина n – количество запросов выходной информации.

3 Определение суммарного времени T выполнения запросов выходной информации по формуле

$$T = \sum_{i=1}^{n} T_i , \qquad (6.3)$$

где T_i – момент времени окончания обслуживания запросов выходной информации;

i – номер запроса выходной информации, $i = \overline{1, n}$;

n – количество запросов выходной информации.

4 Определение $\overline{T}$ среднего времени выполнения запросов выходной информации по формуле

$$\overline{T} = \frac{T}{n}, \qquad (6.4)$$

где T – суммарное время выполнения запросов выходной информации;

n – количество запросов выходной информации.

5 Проверка критерия своевременности представления запрашиваемой выходной информации по формуле

$$\overline{T} \le t_{зад}, \qquad (6.5)$$

где $\overline{T}$ – среднее время выполнения запросов выходной информации;

$t_{зад}$ – заданное время по условию задачи.

6 Вывод результата решения задачи.

Пример решения задачи бесприоритетного обслуживания запросов информации.

Постановка задачи. В информационной системе используются четыре типа запросов A, B, C, D, которые обслуживаются по модели бесприоритетного обслуживания. Каждый из запросов запрашивает один документ. Моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.1. Необходимо определить своевременность представления запрашиваемой информации.

Таблица 6.1 – Исходные данные задачи бесприоритетного обслуживания

Запрос	A	B	C	D
Момент поступления, мин	0	5	8	9
Время обработки, мин	4	6	3	6
Среднее время обработки запроса, мин	14,85			

Решение задачи. 1 Построение диаграммы поступления и обслуживания запросов A, B, C, D (рисунок 6.2).

2 Расчёт T_i моментов времени окончания обслуживания запросов информации A, B, C, D, $i = \overline{1,4}$.

Запрос A. $T_1 = 0 + 4 = 4$ мин. Время ожидания $t_1^{ожид} = 0$ мин. Время обслуживания $t_1^{обсл} = 4$ мин.

Запрос B. $T_2 = 5 + 6 = 11$ мин. Время ожидания $t_2^{ожид} = 1$ мин. Время обслуживания $t_2^{обсл} = 6$ мин.

Запрос C. $T_3 = 8 + (11 - 8) + 3 = 14$ мин. Время ожидания $t_3^{ожид} = 3$ мин. Время обслуживания $t_3^{обсл} = 3$ мин.

Запрос D. $T_4 = 9 + (14 - 9) + 6 = 20$ мин. Время ожидания $t_4^{ожид} = 5$ мин. Время обслуживания $t_4^{обсл} = 6$ мин.

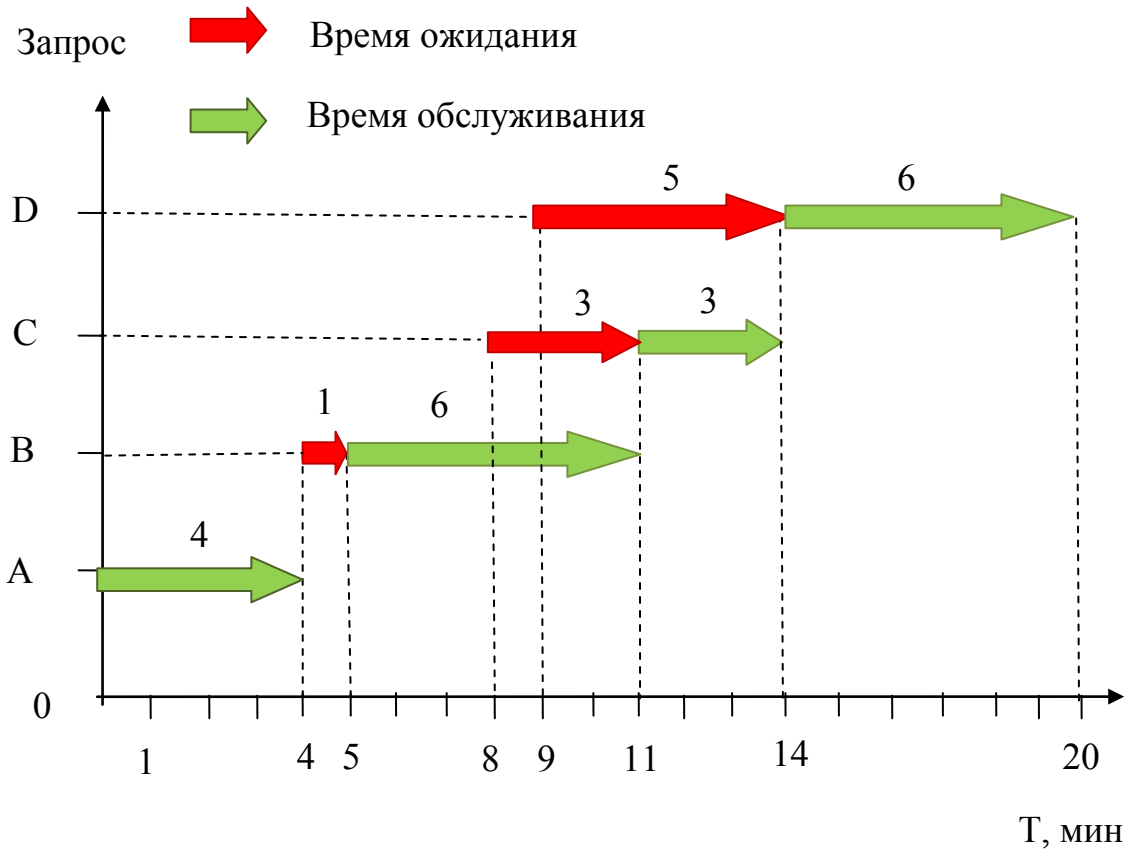


Рисунок 6.2 – Диаграмма поступления и обслуживания запросов информации

Расчётные значения моментов окончания времени обслуживания запросов A, B, C, D приведены в таблице 6.2.

Таблица 6.2 – Моменты окончания обслуживания запросов

Запрос	A	B	C	D
Момент окончания обслуживания, мин	4	11	14	20

3 Расчёт суммарного времени выполнения запросов A, B, C, D по формуле 6.3.

$$T = \sum_{i=1}^{4} T_i = 4 + 11 + 14 + 20 = 49 \text{ мин.}$$

4 Расчёт среднего времени выполнения запросов A, B, C, D по формуле 6.4.

$$\overline{T} = \frac{T}{4} = \frac{49}{4} = 12{,}25 \text{ мин.}$$

5 Проверка условия первого критерия оценки качества информационных систем по формуле 6.5

$$\overline{T} = 12.25 \le 14{,}85 = t_{зад}.$$

Условие первого критерия оценки качества информационных систем выполняется.

6 Вывод результата решения задачи. Информационная система удовлетворяет критерию своевременности представления запрашиваемой информации.

6.5.2 Модель обслуживания запросов информации с относительными приоритетами

Модель обслуживания запросов информации с относительными приоритетами – модель обработки запросов выходных данных с наивысшим приоритетом из очереди запросов. Запрос, который начал выполняться, не прерывается при поступлении запроса с более высоким приоритетом. Приоритеты с меньшим значением имеют больший вес. Приоритет с номером 1 считается наивысшим.

При решении задачи определяется показатель качества – среднее время обработки запроса. Поступление запросов последовательное, по одному запросу каждого типа.

Алгоритм решения задачи обслуживания запросов выходной информации с относительными приоритетами:

1 Построение диаграммы поступления и обслуживания запросов выходной информации с относительными приоритетами.

2 Определение первоочерёдности обслуживания запросов выходной информации в соответствии с моментом поступления и относительным приоритетом запроса в очереди. Обслуживание текущего запроса выходной информации не останавливается, если в процессе выполнения запроса в очередь поступил новый запрос с более высоким приоритетом. Запрос с более высоким приоритетом будет исполняться после завершения обслуживания текущего запроса.

3 Определение T_i моментов времени окончания обслуживания запросов выходной информации с относительными приоритетами. Переменная i – номер запроса выходной информации, $i = \overline{1, n}$. Величина n – количество запросов выходной информации.

4 Определение суммарного времени T выполнения запросов выходной информации по формуле 6.3.

5 Определение $\overline{T}$ среднего времени выполнения запросов выходной информации по формуле 6.4.

6 Проверка критерия своевременности представления запрашиваемой выходной информации по формуле 6.5.

7 Вывод результата решения задачи.

Пример решения задачи обслуживания запросов выходной информации с относительными приоритетами.

Постановка задачи. В информационной системе используются пять типов запросов A, B, C, D, E, которые исполняются по модели обслуживания запросов выходной информации с относительными приоритетами. Приоритет, моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.3. Необходимо определить своевременность представления запрашиваемой информации.

Таблица 6.3 – Исходные данные задачи обслуживания запросов с относительными приоритетами

Запрос	A	B	C	D	E
Приоритет	5	2	1	3	4
Момент поступления, мин	0	3	4	6	7
Время обработки, мин	3	5	4	3	4
Среднее время обработки запроса, мин	11,25				

Решение задачи. 1 Построение диаграммы поступления и обслуживания запросов A, B, C, D, E (рисунок 6.3).

2 Определение первоочерёдности обслуживания запросов выходной информации в соответствии с моментом поступления и относительным приоритетом запроса в очереди.

Запрос A->Запрос B->Запрос C->Запрос D->Запрос E.

3 Определение T_i моментов времени окончания обслуживания запросов выходной информации с относительными приоритетами.

Запрос A. $T_1 = 0 + 3 = 3$ мин. Время ожидания $t_1^{ожид} = 0$ мин. Время обслуживания $t_1^{обсл} = 3$ мин.

Запрос B. $T_2 = 3 + 5 = 8$ мин. Время ожидания $t_2^{ожид} = 0$ мин. Время обслуживания $t_2^{обсл} = 5$ мин.

Запрос C. $T_3 = 4 + (8 - 4) + 4 = 12$ мин. Время ожидания $t_3^{ожид} = 4$ мин. Время обслуживания $t_3^{обсл} = 4$ мин.

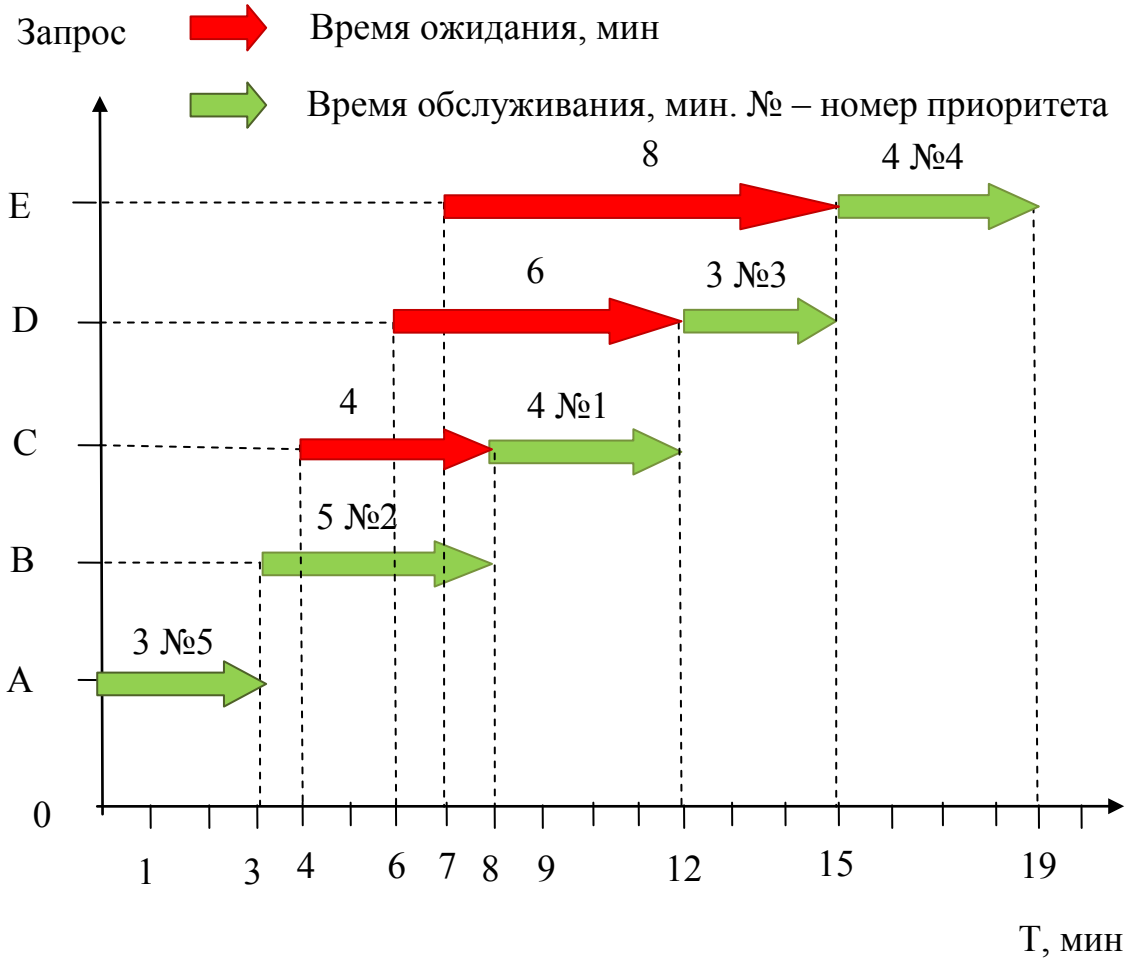


Рисунок 6.3 – Диаграмма поступления и обслуживания запросов информации с относительными приоритетами

Запрос D. $T_4 = 6 + (12 - 6) + 3 = 15$ мин. Время ожидания $t_4^{ожид} = 6$ мин. Время обслуживания $t_4^{обсл} = 3$ мин.

Запрос E. $T5 = 7 + (15 - 7) + 4 = 19$ мин. Время ожидания $t_5^{ожид} = 8$ мин. Время обслуживания $t_5^{обсл} = 4$ мин.

Расчётные значения моментов окончания времени обслуживания запросов A, B, C, D, E приведены в таблице 6.4.

Таблица 6.4 – Моменты окончания обслуживания запросов

Запрос	A	B	C	D	E
Момент окончания обслуживания, мин.	3	8	12	15	19

4 Расчёт суммарного времени выполнения запросов A, B, C, D, E по формуле 6.3.

$$T = \sum_{i=1}^{5} T_i = 3 + 8 + 12 + 15 + 19 = 57 \text{ мин.}$$

5 Расчёт среднего времени выполнения запросов A, B, C, D, E по формуле 6.4.

$$\overline{T} = \frac{T}{5} = \frac{57}{5} = 11{,}4 \text{ мин.}$$

6 Проверка условия первого критерия оценки качества информационных систем по формуле 6.5

$$\overline{T} = 11.4 \geq 11.25 = t_{зад}.$$

Условие первого критерия оценки качества информационных систем не выполняется.

6 Вывод результата решения задачи. Информационная система не удовлетворяет критерию своевременности представления запрашиваемой информации.

6.5.3 Модель обслуживания запросов информации с абсолютными приоритетами

Модель обслуживания запросов информации с абсолютными приоритетами – модель обработки запросов выходных данных, в которой при поступлении нового запроса с более высоким приоритетом выполнение текущего запроса приостанавливается и начинается обслуживание вновь поступившего запроса. Запрос, выполнение которого временно прекращено, возвращается в очередь, и его обслуживание будет продолжено,

когда приоритет запроса окажется наивысшим среди всех запросов информации, находящихся в данный момент в очереди.

При решении задачи определяется показатель качества – среднее время обработки запроса.

Алгоритм решения задачи обслуживания запросов выходной информации с абсолютными приоритетами:

1 Построение диаграммы поступления и обслуживания запросов выходной информации с абсолютными приоритетами.

2 Определение первоочерёдности обслуживания запросов выходной информации в соответствии с моментом поступления и абсолютным приоритетом запроса в очереди. Обслуживание текущего запроса выходной информации приостанавливается, если в процессе выполнения запроса в очередь поступил новый запрос с более высоким приоритетом.

3 Определение времени распределения обработки запросов выходной информации.

4 Определение T_i моментов времени окончания обслуживания запросов выходной информации с абсолютными приоритетами. Переменная i – номер запроса выходной информации, $i = \overline{1, n}$. Величина n – количество запросов выходной информации.

5 Определение суммарного времени T выполнения запросов выходной информации по формуле 6.3.

6 Определение $\overline{T}$ среднего времени выполнения запросов выходной информации по формуле 6.4.

7 Проверка критерия своевременности представления запрашиваемой выходной информации по формуле 6.5.

8 Вывод результата решения задачи.

Пример решения задачи обслуживания запросов выходной информации с абсолютными приоритетами.

Постановка задачи. В информационной системе используются пять типов запросов A, B, C, D, E. Запросы выполняются по модели обслуживания с абсолютными приоритетами. Каждый запрос запрашивает только один документ. Приоритеты, моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.5. Необходимо определить своевременность представления запрашиваемой информации.

Решение задачи. 1 Построение диаграммы поступления и обслуживания запросов A, B, C, D, E с абсолютными приоритетами (рисунок 6.4).

2 Определение первоочерёдности обслуживания запросов выходной информации в соответствии с моментом поступления и абсолютным приоритетом запроса в очереди.

Запрос A->Пустая очередь (1 мин)->Запрос B (1 мин)->Запрос C->Запрос B (4 мин)->Запрос E->Запрос D.

Таблица 6.5 – Исходные данные задачи обслуживания запросов с абсолютными приоритетами

Запрос	A	B	C	D	E
Приоритет	5	2	1	4	3
Момент поступления, мин	0	3	4	6	7
Время обработки, мин	2	5	4	4	4
Среднее время обработки запроса, мин	12,25				

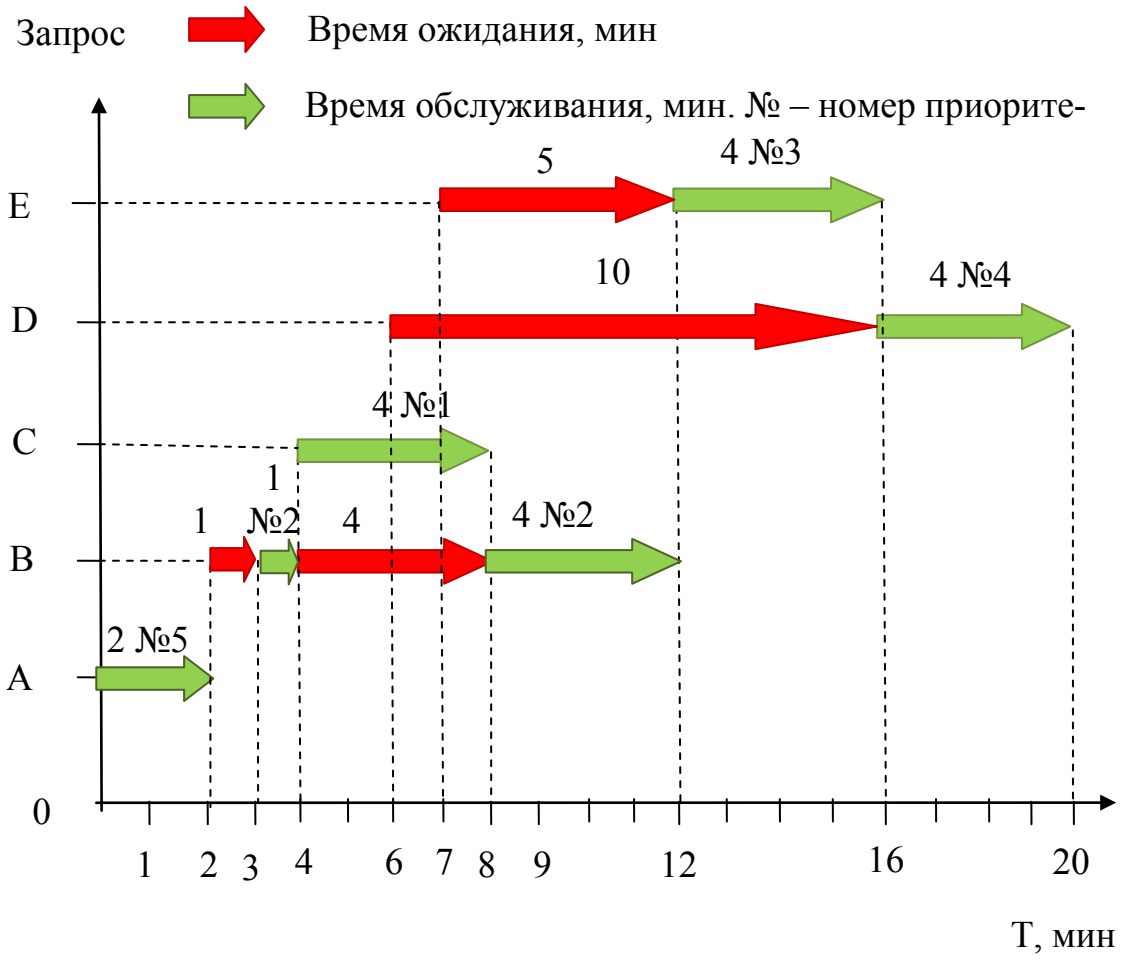


Рисунок 6.4 – Диаграмма поступления и обслуживания запросов информации с абсолютными приоритетами

3 Распределение времени обработки запросов выходной информации (таблица 6.6).

Таблица 6.6 – Распределение времени обработки запросов

Время обработки, мин	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Запрос	A	A	-	B	C	C	C	C	B	B	B	B	E	E	E	E	D	D	D	D

4 Определение T_i моментов времени окончания обслуживания запросов выходной информации с абсолютными приоритетами.

Запрос A. $T_1 = 0 + 2 = 2$ мин. Время ожидания $t_1^{ожид} = 0$ мин. Время обслуживания $t_1^{обсл} = 2$ мин.

Время ожидания поступления запросов (пустая очередь) 1 мин.

Запрос B. $T_2 = 3 + 1 + 4 + 4 = 12$ мин. Время ожидания $t_2^{ожид} = 4$ мин. Время обслуживания $t_2^{обсл} = 4$ мин.

Запрос C. $T_3 = 4 + 4 = 8$ мин. Время ожидания $t_3^{ожид} = 0$ мин. Время обслуживания $t_3^{обсл} = 4$ мин.

Запрос D. $T_4 = 6 + 10 + 4 = 20$ мин. Время ожидания $t_4^{ожид} = 10$ мин. Время обслуживания $t_4^{обсл} = 4$ мин.

Запрос E. $T_5 = 7 + 5 + 4 = 16$ мин. Время ожидания $t_5^{ожид} = 5$ мин. Время обслуживания $t_5^{обсл} = 4$ мин.

Расчётные значения моментов окончания времени обслуживания запросов A, B, C, D, E приведены в таблице 6.7.

Таблица 6.7 – Моменты окончания обслуживания запросов

Запрос	A	B	C	D	E
Момент окончания обслуживания, мин	2	12	8	20	16

5 Расчёт суммарного времени выполнения запросов A, B, C, D, E по формуле 6.3

$$T = \sum_{i=1}^{5} T_i = 2 + 12 + 8 + 20 + 16 = 58 \text{ мин.}$$

6 Расчёт среднего времени выполнения запросов A, B, C, D, E по формуле 6.4

$$\overline{T} = \frac{T}{5} = \frac{58}{5} = 11.6 \text{ мин.}$$

7 Проверка условия первого критерия оценки качества информационных систем по формуле 6.5

$$\overline{T} = 11.6 \le 12.25 = t_{зад}.$$

Условие первого критерия оценки качества информационных систем выполняется.

6 Вывод результата решения задачи. Информационная система удовлетворяет критерию своевременности представления запрашиваемой информации.

Контрольные вопросы

1 Что называется сертификацией соответствия?

2 В чём заключается цель сертификации технологий проектирования и производства систем и программных средств?

3 Какие виды сертификации существуют?

4 Какие требования предъявляются к качеству программных продуктов?

5 Какие используются методики оценки качества программных средств при сертификации?

6 Какие используются критерии оценки качества программно-технических средств?

7 Какие применяются модели обслуживания запросов выходной информации?

8 Какой используется алгоритм решения задачи бесприоритетного обслуживания запросов выходной информации?

9 Какой используется алгоритм решения задачи обслуживания запросов выходной информации с относительными приоритетами?

10 Какой используется алгоритм решения задачи обслуживания запросов выходной информации с абсолютными приоритетами?

Задания для самостоятельной работы

1 В информационной системе используются четыре типа запросов A, B, C, D, которые обслуживаются по модели бесприоритетного обслуживания. Каждый из запросов запрашивает один документ. Моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.8.

Таблица 6.8 – Исходные данные задачи бесприоритетного обслуживания

Запрос	A	B	C	D
Момент поступления, мин	0	4	6	8
Время обработки, мин	3	4	7	5
Среднее время обработки запроса, мин	15,25			

Необходимо определить своевременность представления запрашиваемой информации.

Измените порядок поступления запросов на обратный и определите своевременность представления запрашиваемой информации.

2 В информационной системе используются пять типов запросов A, B, C, D, E, которые исполняются по модели обслуживания запросов выходной информации с относительными приоритетами. Приоритет, моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.9. Необходимо определить своевременность представления запрашиваемой информации.

Таблица 6.9 – Исходные данные задачи обслуживания запросов с относительными приоритетами

Запрос	A	B	C	D	E
Приоритет	4	3	5	1	2
Момент поступления, мин	0	4	5	7	6
Время обработки, мин	5	3	6	2	8
Среднее время обработки запроса, мин	16,75				

Измените порядок поступления запросов на обратный и определите своевременность представления запрашиваемой информации.

3 В информационной системе используются пять типов запросов A, B, C, D, E, которые исполняются по модели обслуживания запросов выходной информации с относительными приоритетами. Приоритет, моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.10. Необходимо определить своевременность представления запрашиваемой информации.

Измените порядок поступления запросов на обратный и определите своевременность представления запрашиваемой информации.

4 В информационной системе используются пять типов запросов A, B, C, D, E. Запросы выполняются по модели обслуживания с абсолютными приоритетами. Каждый запрос запрашивает только один документ. Приоритеты, моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.11. Необходимо определить своевременность представления запрашиваемой информации.

Таблица 6.10 – Исходные данные задачи обслуживания запросов с относительными приоритетами

Запрос	A	B	C	D	E
Приоритет	3	5	2	4	1
Момент поступления, мин	0	8	5	3	6
Время обработки, мин	6	4	7	5	8
Среднее время обработки запроса, мин	14,55				

Таблица 6.11 – Исходные данные задачи обслуживания запросов с абсолютными приоритетами

Запрос	A	B	C	D	E
Приоритет	4	5	2	1	3
Момент поступления, мин	0	6	3	5	2
Время обработки, мин	5	4	8	5	6
Среднее время обработки запроса, мин	17,45				

Измените порядок поступления запросов на обратный и определите своевременность представления запрашиваемой информации.

5 В информационной системе используются пять типов запросов A, B, C, D, E. Запросы выполняются по модели обслуживания с абсолютными приоритетами. Каждый запрос запрашивает только один документ. Приоритеты, моменты поступления запросов, время обработки и среднее время обработки запроса в минутах приведены в таблице 6.12. Необходимо определить своевременность представления запрашиваемой информации.

Таблица 6.12 – Исходные данные задачи обслуживания запросов с абсолютными приоритетами

Запрос	A	B	C	D	E
Приоритет	2	5	3	1	4
Момент поступления, мин	0	8	2	5	3
Время обработки, мин	5	8	6	3	9
Среднее время обработки запроса, мин	18,45				

Измените порядок поступления запросов на обратный и определите своевременность представления запрашиваемой информации.

ЗАКЛЮЧЕНИЕ

Информационные технологии являются основным элементом современного общества. Уровень проникновения ЭВМ и интернета в обществе высокий. Информационная система включает аппаратное и программное обеспечение. Программное обеспечение представляет совокупность программ системы обработки информации и программных документов. Программное обеспечение – вид обеспечения вычислительной системы наряду с аппаратным, математическим, информационным, лингвистическим, организационным и методическим обеспечением. Программное обеспечение реализует возможности вычислительного комплекса и взаимодействует с пользователем, поэтому к нему предъявляются жёсткие требования в отношении качества. На современном этапе развития информационных систем программное обеспечение становится более сложным и ценным. Стоимость его достигает 30-90% стоимости информационных систем. Возрастающая сложность программного обеспечения приводит к увеличению количества ошибок в программном коде, что отрицательно сказывается на результатах его работы. Обеспечение качества программного продукта является актуальной задачей на современном этапе развития информационных технологий.

Верификация и аттестация программного обеспечения осуществляют контроль качества программ и обнаружение ошибок в программном коде. Задачи верификации в рамках жизненного цикла программного обеспечения – выявление дефектов программы, контроль и оценка качества программного обеспечения, предоставление информации о текущем состоянии проекта и характеристиках результатов, выявление наиболее критичных и подверженных ошибкам частей системы.

Методы верификации программного обеспечения – экспертиза, статический анализ, формальные методы, динамические методы, синтетические методы.

В учебном пособии рассматриваются методы оценки качества программного обеспечения, жизненный цикл программных средств и его процессы, описываются оценки характеристик программ на основе лексического анализа, структурной сложности программ, характеристик программ на основе процедурно-ориентированных и объектно-ориентированных метрик. Подробно представляются модели надёжности программных средств: динамические, статические и эвристические. В последних разделах учебного пособия рассматриваются теоретические положения по стандартизации и сертификации качества программного обеспечения и критериям оценок качества.

Для закрепления теоретических знаний и приобретения практических навыков в оценке качества программных средств в учебном пособии приводятся вопросы и варианты заданий самостоятельной работы.

СПИСОК ЛИТЕРАТУРЫ

1 Сандлер К., Баджет Т., Майерс Г. Искусство тестирования программ. – Москва : Вильямс, 2015. – 272 с.

2 Ананьева Т. Н., Исаев Г. Н., Новикова Н. Г. Стандартизация, сертификация и управление качеством программного обеспечения : учебное пособие. – Москва : Инфра-М, 2016. – 232 с.

3 Черников Б. В., Управление качеством программного обеспечения : учебник. – Москва : Форум, 2012. – 240 с.

4 Поклонов Б. Е. Оценка качества программного обеспечения. – Москва : Форум, 2012. – 400 с.

5 Чекал Е. Г. Надёжность информационных систем : учебное пособие : в 2 ч. – Ульяновск : УлГУ, 2012. – Ч.1. – 118 с.

6 Благодатских В. А., Волнин В. А., Поскакалов К. Ф. Стандартизация разработки программных средств : учеб. пособие / под ред. О. С. Разумова. – Москва : Финансы и статистика, 2005. – 288 с.

7 Гагарина Л. Г., Кокорева Е. В., Виснадул Б. Д. Технология разработки программного обеспечения : учебное пособие / под ред. Л. Г. Гагариной. – Москва : ИД ФОРУМ: ИНФРА-М, 2008. – 400 с.

8 Налютин Н. Ю., Синицын С. В. Верификация программного обеспечения. – Москва : МИФИ, 2006. – 157 с.

9 Гурин Р .Е., Рудаков И. В., Ребриков А. В. Методы верификации программного обеспечения // Инженерный вестник. – 2015. – №9. – С. 549-562.

10 Кулямин В. В. Методы верификации программного обеспечения. URL: http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf.

11 Модели качества и надёжности в программной инженерии. URL: http://www.intuit.ru/studies/courses/2190/237/lecture/6136?page=1.

12 Черников Б. В., Поклонов Б. Е. Оценка качества программного обеспечения: Практикум : учебное пособие. – Москва : ФОРУМ : ИНФРА-М, 2012. – 400 с.

13. Решка Дж., Пол Дж., Дастин Э. Тестирование программного обеспечения. Внедрение, управление и автоматизация. – Москва : Лори, 2012. – 600 с.

14. Сандлер К., Баджет Т., Майерс Г. Искусство тестирования программ. – Москва : Вильямс, 2012. – 272 с.

15. Канер С., Фолк Дж. Тестирование программного обеспечения. – Москва : ДиаСофт, 2001. – 538 с.

16. Котляров В. П. Основы тестирования программного обеспечения. – Москва : Интернет Университет Информационных Технологий ; БИНОМ. Лаборатория знаний, 2006. – 285 с.

17 Майерс Г. Дж. Искусство тестирования программ. – Москва : Финансы и статистика, 1982. – 176 с.

18. Майерс Г. Дж. Надежность программного обеспечения. – Москва : Мир, 1980. – 360 с.

19. Анашкина Н. В. Технологии и методы программирования. – Москва: Издательский центр «Академия», 2012. — 384 с.

20. Макконнелл С. Совершенный код. – Санкт-Петербург : Питер, 2005. – 896 с.

21. Бейзер Б. Тестирование черного ящика. – Санкт-Петербург : Питер, 2005. – 318 с.

22. Брауде Э. Технология разработки программного обеспечения. – Санкт-Петербург : Питер, 2004. – 655 с.

23. Степанченко И. В. Методы тестирования программного обеспечения : учебное пособие. – Волгоград : ВолгГТУ, 2006. – 74 с. URL: http://window.edu.ru/resource/765/45765(дата обращения: 06.03.2017).

24 Предеина О. Ю. Метрология, стандартизация и сертификация программного обеспечения : учебное пособие. – Курган : Изд-во КГУ, 2004. – 87 с.

25 ГОСТ 28195-89. Оценка качества программных средств. Общие положения (утв. Постановлением Госстандарта СССР от 28.07.1989 №2507).

26 Липаев В. В. Сертификация программных средств. – Москва : СИНТЕГ, 2010. – 348 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Международные и государственные стандарты, регламентирующие требования, жизненный цикл, испытания и сертификацию комплексов программ

1 **CMMI – Capability Maturity Model Integration for Product and Process Development** – Интегрированная модель оценивания зрелости продуктов и процессов разработки программных средств.

2 **ISO 15288:2002**. Системная инженерия. Процессы жизненного цикла систем.

3 **ISO 19760:2003**. Системная инженерия. Руководство по применению стандарта ISO 15288:2002.

4 **ISO 12207:2008**. ИТ. Процессы жизненного цикла программных средств.

5 **ISO 15271:1998**. (ГОСТ Р – 2002). ИТ. Руководство по применению ISO 12207:2008.

6 **ISO 16326:1999**. (ГОСТР – 2002). ИТ. Руководство по применению ISO 12207:2008 при административном управлении проектами.

7 **ISO 15504:1-5:2003-2006**. ИТ. Процесс аттестации. Ч.1. Концепция и словарь. Ч.2. Выполнение аттестации. Ч.3 Руководство по производству аттестации. Ч. 4. Руководство пользователей для процессов усовершенствования и определения зрелости процессов. Ч. 5. Образец модели процессов аттестации.

8 **ГОСТ Р 51904 – 2002**. Программное обеспечение встроенных систем. Общие требования к разработке и документированию.

9 **ISO 9000:2000**. (ГОСТ Р – 2001). Программное обеспечение встроенных систем. Общие требования к разработке и документированию.

10 **ISO 9001:2000**. (ГОСТ Р – 2001). Система менеджмента (административного управления) качества. Требования.

11 **ISO 90004:2000**. (UJCN H – 2001). Система менеджмента (административного управления) качества. Руководство по улучшению деятельности.

12 **ISO 9003:2004**. Руководство по применению стандарта ISO 9001:2000 к программным средствам.

13 **ГОСТ Р 40.003:2005**. Порядок сертификации систем менеджмента качества на соответствие ГОСТ Р ИСО 9001 – 2001.

14 **Система сертификации ГОСТ Р.** Временный порядок сертификации производств с учётом требований ГОСТ Р ИСО 9001 – 2001.

15 **ГОСТ Р ИСО 19011Ж2003**. Руководящие указания по аудиту систем менеджмента качества.

16 **ISO 10005:2005**. (ГОСТ Р – 2007). Менеджмент организации (Административное управление качеством). Руководящие указания по планированию качества.

17 **ISO 10006:1997**. (ГОСТ) Руководство по качеству при управлении проектом.

18 **ISO 10007:2007**. (ГОСТ Р – 2007). Административное управление качеством. Руководящие указания при управлении конфигурацией.

19 **ISO 10013:2001**. (ГОСТ Р – 2007). Менеджмент организации. (Административное управление качеством). Руководство по документированию систем менеджмента качества.

20 **ISO 12182:1998**. (ГОСТ Р – 2002). ИТ. Классификация программных средств.

21 **ISO 9126:1991**. (ГОСТ – 1993). ИТ. Оценка программного продукта. Характеристики качества и руководство по их применению.

22 **ISO 9126-1-4: 2002**. ИТ. ТО. Качество программных средств: Ч.1. Модель качества.Ч.2. Внешние метрики. Ч. 3. Внутренние метрики. Ч. 4. Метрики качества в использовании.

23 **ISO 14598-1-6:1998-2000**. Оценивание программного продукта. Ч.1. Общий обзор. Ч. 2. Планирование и управление. Ч. 3. Процессы для разработчиков. Ч. 4. Процессы для покупателей. Ч. 5. Процессы для оценщиков. Ч. 6. Документирование и оценивание модулей.

24 **ISO 25000:2005**. ТО. Руководство для применения новой серии стандартов по качеству программных средств на базе обобщения стандартов ISO 9126:1-4:2002 и ISO 14598:1-6:1998-2000.

25 **ISO 15939:2002**. Процесс измерения программных средств.

26 **IEC 61508:1-6: 1998-2000**. Функциональная безопасность электрических / электронных и программируемых электронных систем. Часть 3. Требования к программному обеспечению. Часть 6. Руководство по применению стандартов IEC 61508-2 и IEC 61508-3.

27 **ISO 15408-1-3. 1999.** (ГОСТ Р – 2002). Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий.
Ч. 1. Введение и общая модель. Ч. 2. Защита функциональных требований. Ч.3. Защита требований к качеству.

28 **ISO 13335-1-5. 1996-1998**.ИТ. ТО. Руководство по управлению безопасностью. Ч.1. Концепция и модели обеспечения безопасности информационных технологий. Ч. 2. Планирование и управление безопасностью ИТ. Ч. 4. Селекция (выбор) средств обеспечения безопасности. Ч. 5. Безопасность внешних связей.

29 **ISO 10181:1-7. BOC. 1996-1998**. Структура работ по безопасности в открытых системах. Ч. 1. Обзор. Ч. 2. Структура работ по аутентификации. Ч. 3. Структура работ по управлению доступом. Ч. 4. Структура работ по безотказности. Ч. 6. Структура работ по обеспечению целостности. Ч. 7. Структура работ по проведению аудита на безопасность.

30 **ISO 14252:1996**. (IEEE 1003.0). Руководство по функциональной среде открытых систем POSIX.

31 **ISO 9945:1-4:2003**. ИТ. Интерфейсы переносимых операционных систем. Ч. 1. Базовые определения. Ч. 2. Системные интерфейсы. Ч. 3. Команды управления и сервисные программы. Ч. 4. Обоснование.

32 **ISO 13210:1994**. ИТ. Методы тестирования для измерения соответствия стандарта POSIX.

33 **ISO 14756:1999**. ИТ. Измерение и оценивание производительности программных средств компьютерных вычислительных систем.

34 **12119:1994**. (ГОСТ Р – 2000). ИТ. Требования к качеству и тестирование.

35 **ISO 14764:1999**. (ГОСТ Р – 2002). ИТ. Сопровождение программных средств.

36 **ISO 15846:1998**. ТО. Процессы жизненного цикла программных средств. Конфигурационное управление программными средствами.

37 **ISO 16085:2004**. Характеристики процессов управления рисками при разработке, применении и сопровождении программных средств.

38 **ISO 6592:2000** ОИ. Руководство по документации для вычислительных систем.

39 **ISO 9294:1990** (ГОСТ – 1993). ТО. ИТ. Руководство по управлению документированием программного обеспечения.

40 **ISO 9127:1990** (ГОСТ– 1993). ТО.ИТ. Руководство по управлению документированием программного обеспечения.

41 **ISO 15910:1999** (ГОСТ Р – 2002) ИТ. Пользовательская документация программных средств.

42 **ISO 18019:2004** ИТ. Руководство по разработке пользовательской документации на прикладные программные средства для офисов, бизнеса и профессиональных применений.

43 **ГОСТ Р 51901-2002**. Управление надёжностью. Анализ риска технологических систем.

44 **DO-178 B-1995**. Соглашение по сертификации бортовых систем и оборудования в части программного обеспечения.

ПРИЛОЖЕНИЕ Б

Состав и соответствие показателей качества программных систем на различных фазах жизненного цикла в соответствии с ГОСТ 28195-89

Таблица Б.1 — Показатели качества программных средств

Факторы	Критерии	Метрики	Фазы жизненного цикла					
			Анализ	Проектирование	Реализация	Тестирование	Изготовление	Сопровождение
Надёжность	Устойчивость функционирования	Средства восстановления при ошибках на входе	+	+	+	+	+	+
		Средства восстановления при сбоях оборудования	+	+	+	+	+	+
		Реализация управления средствами восстановления		+	+	+	+	+
	Работоспособность	Функционирование в заданных режимах			+	+	+	+
		Обеспечение обработки заданного объёма информации			+	+	+	+
Сопровождаемость	Простота конструкции	Простота архитектуры проекта	+	+				+
		Сложность архитектуры проекта		+	+	+	+	+
		Межмодульные связи		+				
		Простота кодирования			+	+	+	
	Наглядность	Комментарии логики программного проекта			+	+	+	+
		Оформление текста программ			+	+	+	+
	Структурность	Соблюдение принципа нисходящего программирования			+	+	+	+

Продолжение таблицы Б.1

Удобство применения	Лёгкость освоения	Освоение работы программного обеспечения					+	+
		Документация для освоения					+	+
	Доступность эксплуатационных документов	Полнота документации			+	+	+	+
		Точность документации			+	+	+	+
		Понятность документации			+	+	+	+
		Техническое исполнение документации			+	+	+	+
		Прослеживание вариантов документации			+	+	+	+
	Удобство эксплуатации и обслуживания	Эксплуатация	+	+	+	+	+	+
		Управление меню	+	+	+	+	+	+
		Функция поддержки справочной системы (помощи)	+	+	+	+	+	+
		Управление данными	+	+	+	+	+	+
		Рабочие процедуры	+	+	+	+	+	+
Эффективность	Уровень автоматизации	Функции автоматизации	+	+	+	+	+	+
	Временная эффективность	Затраты времени	+		+	+	+	+
	Ресурсоёмкость	Использование вычислительных ресурсов	+	+	+	+	+	+
Универсальность	Гибкость	Широта охвата функций	+	+	+	+	+	+
		Простота архитектуры проекта		+	+	+	+	+
		Сложность архитектуры проекта		+	+	+	+	+
		Сложность структуры кода программы			+	+	+	+
		Применение стандартных протоколов связи			+	+	+	+
		Применение стандартных интерфейсных программ			+	+	+	+

Продолжение таблицы Б.1

Универсальность	Мобильность	Зависимость от используемого комплекса технических средств	+		+	+	+	+
		Зависимость от базового программного обеспечения	+		+	+	+	+
		Изоляция немобильности	+		+	+	+	+
	Модифицируемость	Простота кодирования			+	+	+	+
		Число комментариев			+	+	+	+
		Качество комментариев			+	+	+	+
		Использование описательных средств языка			+	+	+	+
		Независимость модулей			+	+	+	+
Корректность	Полнота реализации	Полнота документации разработчика	+	+	+	+	+	+
		Полнота программной документации	+		+	+	+	+
	Согласованность	Непротиворечивость документации			+	+	+	+
		Непротиворечивость программы	+	+	+	+	+	
		Единообразие межмодульных и пользовательских интерфейсов	+	+	+	+	+	+
		Единообразие кодирования и определения переменных	+	+	+	+	+	+
		Соответствие документации стандартам программирования		+	+	+	+	
	Проверенность	Требования к полноте тестирования	+		+	+	+	+

ПРИЛОЖЕНИЕ В

Перечень оценочных элементов

Таблица В.1 – Состав метрик фактора «Надёжность»

Метрики	Наименование оценочных элементов	Метод оценки	Возможные значения
1 Средства восстановления при ошибке на входе	1 Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных	Экспертный	0…1
	2 Возможность обработки ошибочных ситуаций	Экспертный	0…1
	3 Полнота обработки ошибочных ситуаций	Экспертный	0…1
	4 Наличие тестов для проверки допустимых значений входных данных	Экспертный	0…1
	5 Наличие системы контроля полноты входных данных	Экспертный	0…1
	6 Наличие средств контроля корректности входных данных	Экспертный	0…1
	7 Наличие средств контроля непротиворечивости входных данных	Экспертный	0…1
	8 Наличие проверки параметров и адресов по диапазону их значений	Экспертный	0…1

Продолжение таблицы В.1

	9 Наличие обработки граничных результатов	Экспертный	0…1
	10 Наличие обработки неопределённостей	Экспертный	0…1
2 Средства восстановления при сбоях оборудования	1 Наличие требований к программе по восстановлению процесса выполнения в случае сбоя операционной системы, процессора, внешних устройств	Экспертный	0…1
	2 Наличие требований к программе по восстановлению результатов при отказах процессора	Экспертный	0…1
	3 Наличие средств восстановления процесса в случае сбоев оборудования	Экспертный	0…1
	4 Наличие возможности разделения по времени выполнения отдельных функций программ	Экспертный	0…1
	5 Наличие возможности повторного старта с точки останова	Экспертный	0…1
3 Реализация средствами восстановления	1 Наличие централизованного управления процессами, конкурирующими из-за ресурсов	Экспертный	0…1

Продолжение таблицы В.1

	2 Наличие возможности обходить ошибочные ситуации в процессе вычисления	Экспертный	0…1
	3 Наличие средств, обеспечивающих завершение процесса решения в случае помех	Экспертный	0…1
	4 Наличие средств, обеспечивающих выполнение программы в сокращённом объёме в случае ошибок или помех	Экспертный	0…1
	5 Показатель устойчивости к искажающим воздействиям	Расчётный	$P = 1 - \frac{D}{K}$, где D – число экспериментов, в которых искажающие воздействия приводят к отказу; K – число экспериментов, в которых имитировались искажающие воздействия
4 Функционирование в заданных режимах	1 Вероятность безотказной работы	Расчётный	$P = 1 - Q/N$, где Q – число зарегистрированных отказов; N – число экспериментов
5 Обеспечение обработки заданного объёма информации	1 Оценка по среднему времени восстановления	Расчётный	$Q = \begin{cases} 1, \text{если } T_в < T_в^{доп}; \\ \dfrac{T_в^{доп}}{T_в}, \text{если } T_в > T_в^{доп}, \end{cases}$ где $T_в$ – среднее время восстановления; $T_в^{доп}$ – допустимое среднее время восстановления

Продолжение таблицы В.1

	2 Оценка по продолжительности преобразования входного набора данных в выходной	Расчётный	$Q_{ni} = \begin{cases} 1, \text{если } T_{ni} < T_{ni}^{доп}; \\ \dfrac{T_{ni}^{доп}}{T_{ni}}, \\ \text{если } T_{ni} > T_{ni}^{доп}, \end{cases}$ где $T_{ni}^{доп}$ – допустимое время преобразования i-го входного набора данных; T_{ni} – фактическое время преобразования входного набора данных

Таблица В.2 – Состав метрик фактора «Сопровождаемость»

Метрики	Наименование оценочных элементов	Метод оценки	Возможные значения
1 Простота архитектуры проекта	1 Наличие модульной схемы программы	Экспертный	0…1
	2 Оценка программы по числу уникальных модулей	Экспертный	0…1
2 Сложность архитектуры проекта	1 Наличие ограничений на размеры модулей	Экспертный	0…1
3 Межмодульные связи	1 Наличие требований к независимости модулей программы от типов и форматов выходных данных	Экспертный	0…1

Продолжение таблицы В.2

	2 Наличие проверки корректности передаваемых данных	Экспертный	0…1
	3 Оценка простоты программы по числу точек входа и выхода	Расчётный	$W = \frac{1}{(D+1)*(F+1)}$, где D – общее число точек входа в программу; F – общее число точек выхода из программы
	4 Осуществляется ли передача результатов работы модуля через вызывающий его модуль	Экспертный	0…1
	5 Осуществляется ли контроль за правильностью данных, поступающих в вызывающий модуль от вызываемого модуля	Экспертный	0…1
4 Соблюдение принципа нисходящего программирования	1 Использование при построении программ метода структурного программирования	Экспертный	0…1
	2 Соблюдение принципа разработки программы сверху вниз	Экспертный	0…1
	3 Оценка программы по числу циклов с одним входом и одним выходом	Экспертный	0…1
	4 Оценка программы по числу циклов	Экспертный	0…1
5 Комментарии логики программного проекта	1 Наличие комментариев ко всем машинозависимым частям программы	Экспертный	0…1

Продолжение таблицы В.2

	2 Наличие комментариев к машинозависимым операторам программы	Экспертный	0…1
	3 Наличие комментариев в точках входа и выхода программы	Экспертный	0…1
6 Оформление текста программ	1 Сопровождение комментариев принятым соглашением	Экспертный	0…1
	2 Наличие комментариев-заголовков программы с указанием её структурных и функциональных характеристик	Экспертный	0…1
	3 Оценка ясности и точности описания последовательности функционирования всех элементов программы	Экспертный	0…1
7 Простота кодирования	1 Используется ли язык высокого уровня	Экспертный	0…1
	2 Оценка простоты программы по числу переходов по условию	Расчётный	$U = 1 - \frac{A}{B}$, где A – общее число переходов по условию; B – общее число исполняемых операторов

Таблица В.3 – Состав метрик фактора «Удобство применения»

Метрики	Наименование оценочных элементов	Метод оценки	Возможные значения
1 Освоение работы программного обеспечения	1 Возможность освоения программных средств по документации	Экспертный	0…1

Продолжение таблицы В.3

	2 Возможность освоения программных средств на контрольном примере при помощи ЭВМ	Экспертный	0…1
	3 Возможность поэтапного освоения	Экспертный	0…1
2 Документация для освоения	1 Полнота и понятность документации для освоения	Экспертный	0…1
	2 Точность документации для освоения	Экспертный	0…1
	3 Техническое освоение документации	Экспертный	0…1
3 Полнота документации	1 Наличие краткой аннотации	Экспертный	0…1
	2 Наличие описания решаемых задач	Экспертный	0…1
	3 Наличие описания структуры функций программы	Экспертный	0…1
	4 Наличие описания основных функций программы	Экспертный	0…1
	5 Наличие описания частных функций программы	Экспертный	0…1
	6 Наличие описания алгоритма	Экспертный	0…1
	7 Наличие описания межмодульных интерфейсов	Экспертный	0…1
	8 Наличие описания пользовательских интерфейсов	Экспертный	0…1
	9 Наличие описания входных и выходных данных	Экспертный	0…1
	10 Наличие описаний диагностических сообщений	Экспертный	0…1

Продолжение таблицы В.3

	11 Наличие описания основных характеристик программы	Экспертный	0…1
	12 Наличие описания программной среды функционирования программы	Экспертный	0…1
	13 Достаточность документации для ввода программы в эксплуатацию	Экспертный	0…1
	14 Наличие информации о технологии переноса для мобильных программ	Экспертный	0…1
4 Точность документации	1 Соответствие оглавления содержанию документации	Экспертный	0…1
	2 Оценка оформления документации	Экспертный	0…1
	3 Грамматическая правильность изложения документации	Экспертный	0…1
	4 Отсутствие противоречий в документации	Экспертный	0…1
	5 Отсутствие неправильных ссылок в документации	Экспертный	0…1
	6 Ясность формулировок и описаний в документации	Экспертный	0…1
	7 Отсутствие неоднозначных формулировок и описаний в документации	Экспертный	0…1
	8 Правильность использования терминов в документации	Экспертный	0…1
	9 Краткость, отсутствие лишней детализации в документации	Экспертный	0…1

Продолжение таблицы В.3

	10 Единство формулировок в документации	Экспертный	0…1
	11 Единство обозначений в документации	Экспертный	0…1
	12 Отсутствие ненужных повторений в документации	Экспертный	0…1
	13 Наличие нужных объяснений в документации	Экспертный	0…1
5 Понятность документации	1 Оценка стиля изложения документации	Экспертный	0…1
	2 Дидактическая разделённость документации	Экспертный	0…1
	3 Формальная разделённость	Экспертный	0…1
	4 Ясность логической структуры документации	Экспертный	0…1
	5 Соблюдение стандартов и правил изложения в документации	Экспертный	0…1
	6 Оценка по числу ссылок вперёд в тексте документов	Экспертный	0…1
6 Техническое исполнение документации	1 Наличие оглавления документа	Экспертный	0…1
	2 Наличие предметного указателя документа	Экспертный	0…1
	3 Наличие перекрёстных ссылок в документе	Экспертный	0…1
	4 Наличие всех требуемых разделов	Экспертный	0…1
	5 Соблюдение непрерывности нумерации страниц документов	Экспертный	0…1
	6 Отсутствие незаконченных разделов, абзацев, предложений в документах	Экспертный	0…1

Продолжение таблицы В.3

	7 Наличие всех рисунков, чертежей, формул, таблиц в документах	Экспертный	0…1
	8 Наличие всех строк и примечаний в документе	Экспертный	0…1
	9 Логический порядок частей внутри главы документа	Экспертный	0…1
7 Прослеживание вариантов документации	1 Наличие полного перечня документации	Экспертный	0…1
8 Эксплуатация	1 Уровень языка общения пользователя с программой	Экспертный	0…1
	2 Лёгкость и быстрота загрузки и запуска программы	Экспертный	0…1
	3 Лёгкость и быстрота завершения работы программы	Экспертный	0…1
	4 Возможность распечатки содержимого программы	Экспертный	0…1
	5 Возможность останова и повторного запуска работы без потерь информации	Экспертный	0…1
9 Управление меню	1 Соответствие меню требованиям пользователя	Экспертный	0…1
	2 Возможность прямого перехода вверх и вниз по многоуровневому меню (пропуск уровней)	Экспертный	0…1
10 Функция поддержки справочной системы (помощи)	1 Возможность управления подробностью получаемых выходных данных	Экспертный	0…1
	2 Достаточность полученной информации для продолжения работы	Экспертный	0…1

Продолжение таблицы В.3

11 Управление данными	1 Обеспечение удобства ввода данных	Экспертный	0…1
	2 Лёгкость восприятия	Экспертный	0…1
12 Рабочие процедуры	1 Обеспечение программой выполнения предусмотренных рабочих процедур	Экспертный	0…1
	2 Достаточность информации, выдаваемой программой для составления дополнительных процедур		

Таблица В.4 – Состав метрик фактора «Эффективность»

Метрики	Наименование оценочных элементов	Метод оценки	Возможные значения
1 Широта охвата функций	1 Проблемно-ориентированные функции	Экспертный	0…1
	2 Машинно-ориентированные функции	Экспертный	0…1
	3 Функции ведения и управления	Экспертный	0…1
	4 Функции ввода-вывода	Экспертный	0…1
	5 Функции защиты и проверки данных	Экспертный	0…1
	6 Функции защиты от несанкционированного доступа	Экспертный	0…1
	7 Функции контроля доступа	Экспертный	0…1
	8 Функции защиты от внесения изменений	Экспертный	0…1
	9 Наличие соответствующих границ функциональных областей	Экспертный	0…1
	10 Точность, достигаемая в результатах вычисления	Экспертный	0…1

Продолжение таблицы В.4

2 Затраты времени	1 Время выполнения программ	Экспертный	0…1
	2 Время реакции и ответов	Экспертный	0…1
	3 Время подготовки	Экспертный	0…1
	4 Затраты времени на защиту данных	Экспертный	0…1
	5 Время компиляции	Экспертный	0…1
3 Использование вычислительных процедур	1 Требуемый объём внутренней памяти	Экспертный	0…1
	2 Требуемый объём внешней памяти	Экспертный	0…1
	3 Требуемые периферийные устройства	Экспертный	0…1
	4 Требуемое базовое программное обеспечение	Экспертный	0…1

Таблица В.5 – Состав метрик фактора «Универсальность»

Метрики	Наименование оценочных элементов	Метод оценки	Возможные значения
1 Широта охвата функций	1 Оценка числа потенциальных пользователей	Экспертный	0…1
	2 Оценка числа функций программных средств	Экспертный	0…1
	3 Насколько набор функций удовлетворяет требованиям пользователя	Экспертный	0…1
	4 Насколько возможности программ охватывают область решаемых пользователем задач	Экспертный	0…1
	5 Возможность настройки формата входных данных для конкретных пользователей	Экспертный	0…1
2 Простота архитектуры проекта	1 Наличие схемы иерархии модулей программ	Экспертный	0…1

Продолжение таблицы В.5

	2 Оценка независимости модулей	Экспертный	0…1
	3 Оценка числа уникальных элементов	Экспертный	0…1
	4 Используется ли в текущем вызове модуля информация, полученная в предыдущем вызове	Экспертный	0…1
	5 Оценка организации точек входа и выхода модуля	Экспертный	0…1
	6 Наличие описания атрибутов модуля	Экспертный	0…1
3 Сложность Архитектуры проекта	1 Оценка программ по числу переходов и точек ветвления	Экспертный	0…1
4 Сложность структуры кода программы	1 Использование метода пошагового уточнения	Экспертный	0…1
	2 Наличие описания структуры программ	Экспертный	0…1
	3 Наличие описания связей между элементами структуры программы	Экспертный	0…1
	4 Наличие в программе повторного выполнения функций (подпрограмм)	Экспертный	0…1
5 Применение стандартных протоколов связи	1 Использование стандартных протоколов связи	Экспертный	0…1
6 Применение стандартных интерфейсных программ	1 Использование стандартных интерфейсных подпрограмм	Экспертный	0…1
7 Зависимость от используемого комплекса технических средств	1 Оценка зависимости программ от ёмкости оперативной памяти		0…1
	2 Оценка зависимости временных характеристик программ от скорости вычислений ЭВМ	Экспертный	0…1

Продолжение таблицы В.5

	3 Оценка зависимости функционирования программы от числа внешних запоминающих устройств и их общей ёмкости	Экспертный	0…1
	4 Оценка зависимости функционирования программы от специальных устройств ввода-вывода	Экспертный	0…1
8 Зависимость от базового программного обеспечения	1 Применение специальных языков программирования	Экспертный	0…1
	2 Оценка зависимости программы от программ операционной системы	Экспертный	0…1
	3 Зависимость от других программных средств	Экспертный	0…1
9 Изоляция немобильности	1 Оценка локализации переносимой части программы	Экспертный	0…1
10 Простота кодирования	1 Оценка использования отрицательных булевых выражений	Экспертный	0…1
	2 Оценка программы по использованию условных переходов	Экспертный	0…1
	3 Оценка программы по использованию безусловных переходов	Экспертный	0…1
	4 Оформление процедур входа и выхода из циклов	Экспертный	0…1
	5 Ограничения на модификацию переменной индексации в цикле	Экспертный	0…1
	6 Оценка модулей по направлению потока управления	Экспертный	0…1

Продолжение таблицы В.5

	7 Оценка программы по использованию локальных переменных	Экспертный	0…1
11 Число комментариев	1 Оценка программы по числу комментариев	Экспертный	0…1
12 Количество комментариев	1 Наличие заголовка в программе	Экспертный	0…1
	2 Комментарии к точкам ветвления	Экспертный	0…1
	3 Комментарии к машинозависимым частям программы	Экспертный	0…1
	4 Комментарии к машинозависимым операторам программы	Экспертный	0…1
	5 Комментарии к операторам объявления переменных	Экспертный	0…1
	6 Оценка семантики операторов	Экспертный	0…1
	7 Наличие соглашений по форме представления комментариев	Экспертный	0…1
	8 Наличие общих комментариев к программам	Экспертный	0…1
13 Использование описательных средств языка	1 Использование языков высокого уровня	Экспертный	0…1
	2 Семантика имён используемых переменных	Экспертный	0…1
	3 Использование отступов, сдвигов и пропусков при формировании текста	Экспертный	0…1
	4 Размещение операторов по строкам	Экспертный	0…1
14 Независимость модулей	1 Передача информации для управления по параметрам	Экспертный	0…1
	2 Параметрическая передача входных данных	Экспертный	0…1

Продолжение таблицы В.5

	3 Наличие передачи результатов работы между модулями	Экспертный	0…1
	4 Наличие проверки правильности данных, получаемых модулями от вызываемого модуля	Экспертный	0…1
	5 Использование общих областей памяти	Экспертный	0…1

Таблица В.6 – Состав метрик фактора «Корректность»

Метрики	Наименование оценочных элементов	Метод оценки	Возможные значения
1 Полнота документации разработчика	1 Наличие всех необходимых документов	Экспертный	0…1
	2 Наличие описания и схемы иерархии модулей программы	Экспертный	0…1
	3 Наличие описания основных функций	Экспертный	0…1
	4 Наличие описания частных функций	Экспертный	0…1
	5 Наличие описания данных	Экспертный	0…1
	6 Наличие описания алгоритма	Экспертный	0…1
	7 Наличие описания интерфейса между модулями	Экспертный	0…1
	8 Наличие описания интерфейсов с пользователями	Экспертный	0…1
	9 Наличие описания используемых числовых методов	Экспертный	0…1
	10 Указаны ли все числовые методы	Экспертный	0…1
	11 Наличие описания всех параметров	Экспертный	0…1

Продолжение таблицы В.6

	12 Наличие описания методов настройки системы	Экспертный	0…1
	13 Наличие описания всех диагностических сообщений	Экспертный	0…1
	14 Наличие описания способов проверки работоспособности программы	Экспертный	0…1
2 Полнота программной документации	1 Реализация всех исходных модулей	Экспертный	0…1
	2 Реализация всех основных функций	Экспертный	0…1
	3 Реализация всех частных функций	Экспертный	0…1
	4 Реализация всех алгоритмов	Экспертный	0…1
	5 Реализация всех возможностей в системе	Экспертный	0…1
	6 Реализация всех интерфейсов между модулями	Экспертный	0…1
	7 Реализация возможности настройки системы	Экспертный	0…1
	8 Реализация диагностики всех граничных и аварийных ситуаций	Экспертный	0…1
	9 Наличие определения всех данных (переменные, индексы, массивы и прочее)	Экспертный	0…1
	10 Наличие интерфейсов с пользователем	Экспертный	0…1
3 Непротиворечивость документации	1 Отсутствие противоречий в описании частных функций	Экспертный	0…1

Продолжение таблицы В.6

	2 Отсутствие противоречий в описании основных функций в разных документах	Экспертный	0…1
	3 Отсутствие противоречий в описании алгоритмов	Экспертный	0…1
	4 Отсутствие противоречий в описании взаимосвязей в системе	Экспертный	0…1
	5 Отсутствие противоречий в описании интерфейсов между модулями	Экспертный	0…1
	6 Отсутствие противоречий в описании интерфейсов с пользователями	Экспертный	0…1
	7 Отсутствие противоречий в описании настройки системы	Экспертный	0…1
	8 Отсутствие противоречий в описании иерархической структуры сообщений	Экспертный	0…1
	9 Отсутствие противоречий в описании диагностических сообщений	Экспертный	0…1
	10 Отсутствие противоречий в описании данных	Экспертный	0…1
4 Непротиворечивость программы	1 Отсутствие противоречий в выполнении основных функций	Экспертный	0…1
	2 Отсутствие противоречий в выполнении частных функций	Экспертный	0…1
	3 Отсутствие противоречий в выполнении алгоритмов	Экспертный	0…1

Продолжение таблицы В.6

	4 Правильность взаимосвязей	Экспертный	0…1
	5 Правильность реализации интерфейса между модулями	Экспертный	0…1
	6 Правильность реализации интерфейса с пользователем	Экспертный	0…1
	7 Отсутствие противоречий в настройке системы	Экспертный	0…1
	8 Отсутствие противоречий в диагностике системы	Экспертный	0…1
	9 Отсутствие противоречий в общих переменных	Экспертный	0…1
5 Единообразие межмодульных и пользовательских интерфейсов	1 Единообразие способов вызова модулей	Экспертный	0…1
	2 Единообразие процедур возврата управления из модулей	Экспертный	0…1
	3 Единообразие способов сохранения информации для возврата	Экспертный	0…1
	4 Единообразие способов восстановления информации для возврата	Экспертный	0…1
	5 Единообразие организации списков передаваемых параметров	Экспертный	0…1
6 Единообразие кодирования и определения переменных	1 Единообразие наименования каждой переменной и константы	Экспертный	0…1
	2 Все ли одинаковые константы встречаются во всех программах под одинаковыми именами	Экспертный	0…1

Продолжение таблицы В.6

	3 Единообразие определения внешних данных во всех программах	Экспертный	0…1
	4 Используются ли разные идентификаторы для разных переменных	Экспертный	0…1
	5 Все ли общие переменные объявлены как общие переменные	Экспертный	0…1
	6 Наличие определений одинаковых атрибутов	Экспертный	0…1
7 Соответствие документации стандартам	1 Комплектность документации в соответствии со стандартами	Экспертный	0…1
	2 Правильное оформление частей документов	Экспертный	0…1
	3 Правильное оформление титульных и заглавных листов документов	Экспертный	0…1
	4 Наличие в документах всех разделов в соответствии со стандартами	Экспертный	0…1
	5 Полнота содержания разделов в соответствии со стандартами	Экспертный	0…1
	6 Деление документов на структурные элементы: разделы, подразделы, пункты, подпункты	Экспертный	0…1
8 Соответствие программы стандартам программирования	1 Соответствие организации вычислительного процесса эксплуатационной документации	Экспертный	0…1

Продолжение таблицы В.6

	2 Правильность заданий на выполнение программы, правильность написания управляющих операторов (отсутствие ошибок)	Экспертный	0…1
	3 Отсутствие ошибок в описании действий пользователя	Экспертный	0…1
	4 Отсутствие ошибок в описании запуска	Экспертный	0…1
	5 Отсутствие ошибок в описании генерации	Экспертный	0…1
	6 Отсутствие ошибок в описании настройки	Экспертный	0…1
9 Требования к полноте тестирования	1 Наличие требований к тестированию программ	Экспертный	0…1
	2 Достаточность требований к тестированию программ	Экспертный	0…1
	3 Отношение числа модулей, отработавших в процессе тестирования и отладок, к общему числу модулей	Расчётный	$\frac{Q_T^M}{Q_0^M}$, где Q_T^M – число модулей, отработавших в процессе тестирования и отладки; Q_0^M – общее число модулей

Продолжение таблицы В.6

	4 Отношение числа логических блоков, отработавших в процессе тестирования и отладок, к общему числу логических блоков в программе	Расчётный	$\frac{Q_T^Б}{Q_0^Б}$, где $Q_T^Б$ – число блоков, отработавших в процессе тестирования и отладки; $Q_0^Б$ – общее число блоков в программе

Учебное издание

Семахин Андрей Михайлович

МЕТОДЫ ВЕРИФИКАЦИИ И ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебное пособие

Редактор Н.М. Быкова

Подписано в печать 28.11.2018	Формат 60x84 1/16	Бумага 80 г/см^2
Печать цифровая	Усл. печ. л. 9,38	Уч.-изд. л. 9,38
Заказ 221	Тираж 35	

БИЦ Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.