Автономная некоммерческая организация дополнительного профессионального образования «Научно-образовательный центр
ВКО «Алмаз – Антей» им. академика В. П. Ефремова»

На правах рукописи

Нарышкин Константин Викторович

МЕТОД ОЦЕНКИ КАЧЕСТВА КОМПЬЮТЕРНЫХ ЭЛЕМЕНТОВ СИСТЕМЫ УПРАВЛЕНИЯ ПРИ ПЕРЕНОСЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА АЛЬТЕРНАТИВНЫЕ АППАРАТНЫЕ ПЛАТФОРМЫ

Специальность 2.3.1 – Системный анализ, управление и обработка информации, статистика

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель:
д.т.н., Новиков Владимир Александрович

Москва – 2024

Оглавление

ВВЕДЕНИЕ

Актуальность темы. Деятельность предприятий (организаций) связана с непрерывным обновлением действующих технических систем. Совершенствование технических систем управления достигается с помощью постоянной разработки и внедрения новых программно-аппаратных комплексов, которые не могут быть разработаны и внедрены единовременно из-за высокой сложности. Создание технологического суверенитета предполагает такой процесс модернизации (в частности, импортозамещение), который позволит за минимальное время заменить действующие программно-аппаратные комплексы в системах управления (компьютерные элементы системы управления) без значительной потери основных характеристик качества.

Основой процесса импортозамещения является создание нового или использование существующего контролируемого аппаратного обеспечения (АО). В процессе модернизации действующих компьютерных элементов (КЭ) создание нового АО связано с расходованием значительного объёма ресурсов без гарантий получения требуемого результата. Использование существующего АО требует меньше ресурсов, но ограничено тем перечнем компонентов, которые доступны для приобретения. После определения перечня доступного АО требуется создать новое или перенести существующее программное обеспечение (ПО) на доступные альтернативные аппаратные платформы (ААП). Как при создании нового ПО, так и при переносе существующего возникает потребность в многокритериальной оценке качества и разработке на этой основе рекомендаций о целесообразности продолжения или прекращения внедрения получаемого программно-аппаратного комплекса.

В научной литературе [1–5] и в руководящих документах [6–9] предлагаются различные методы оценки разрозненных показателей качества. В ситуациях, когда разрабатывается новое аппаратное или программное обеспечение, применяют иерархические модели качества Боэма, МакКола, FURPS, Дроми, Гилба, GQM,

ГОСТ Р ИСО/МЭК 25010-2015. Для оценки показателей качества вычислительных систем используют измерительные, регистрационные, статистические, экспертные методы. В ситуациях, когда используются альтернативные аппаратные платформы для функционирования существующего ПО, возникают трудности в оценке показателей качества. Прежде всего эти трудности связаны с необходимостью учитывать неизвестные аппаратные и программные факторы, в условиях которых функционирует система. Также трудности применения существующих методов связаны с тем, что при объединении имеющегося программного и аппаратного обеспечения пропускается большинство традиционных этапов разработки системы, а, следовательно, для неё характерно отсутствие статистических данных.

Критерии точности (правильность и прецизионность) [6] достигаются за счёт того, что методы оценки показателей качества применяются на всех возможных этапах разработки и внедрения программно-аппаратных комплексов, где существует большой объем статистических данных и незначительное количество неизвестных аппаратно-программных факторов [10]. Модернизация сложных систем управления за счёт переноса ПО на ААП создаёт трудности при оценке показателей качества, что снижает точность оценок, полученных существующими методами.

Степень разработанности темы исследования. Исследованы работы по моделям качества Боэма Б., Щенникова А. Н., Катылюка И. С, Тациана Н. Кудо, по моделям надёжности Балыбердина В. А., Василенко Н. В., Муравьева К. А., по моделям безопасности Ибрагимова Б. Г.

Проведён анализ работ, описывающих измерительные методы программных (Звездин С. В., Лукин В. Н., Каушан В. В., Ермаков М. К.) и аппаратных компонент (Хамадулин Э. Ф.) информационной системы, регистрационные методы, в частности методы тестирования (Коташев А. А., Чупилко М. М., Золотухина Е. Б., Владимиров М. А., Данилов А. Д.), методы анализа дерева отказов (Берман А. Ф.), расчётные методы (Тациана Н. Кудо, Балыбердин В. А., Неборский С. Н., Ковтун Н. И., Билятдинов К. З.).

Проведён анализ применения метода анализа исходного и бинарного кода в следующих процессах: разработка (Лаврищева Е. М., Иванников В. П., Белеванцев А. А., Рыжков Е. А.), обеспечение информационной безопасности (Буйневич М. В., Новиков В. А., Аветисян А. И, Падарян В. А., Израилов К. Е., Каушан В. В., Зегжда П. Д., Еремеев М. А.), проведение судебной экспертизы (Тарасов Д. А.).

Цель исследования. Повышение точности оценки качества сложных систем управления в процессе модернизации, которая обеспечивается переносом существующего программного обеспечения на альтернативные аппаратные платформы за счёт совершенствования моделей и методов оценки качества компьютерных элементов.

Основные задачи исследования:

1. Провести анализ моделей качества и методов оценки качества вычислительных систем в составе систем управления.
2. Разработать модель качества КЭ СУ, показатели которой зависят от параметров исполняемого кода.
3. Разработать алгоритм поиска аппаратно-зависимых ошибок в исполняемом коде, используя сравнительный анализ формальных моделей программных компонент.
4. Разработать методику оценки влияния найденных ошибок в исполняемом коде на характеристики качества, а также комплект программ по реализации разработанной методики.
5. Разработать метод оценки характеристик качества на основе информации об ошибках, полученной в результате анализа исполняемого кода КЭ СУ.
6. Провести экспериментальную проверку полученных результатов исследования. Выработать рекомендации и предложения по использованию анализа исполняемого кода в процессе принятия решений о возможной модернизации КЭ СУ.

Объект исследования. Оценивание качества компьютерных элементов сложной системы управления в процессе модернизации аппаратного и программного обеспечения.

Предмет исследования. Методы и алгоритмы многокритериальной оценки качества компьютерных элементов сложной системы управления в процессе модернизации аппаратного и программного обеспечения.

Научная задача исследования. Разработка научно-методического аппарата, повышающего точность оценки качества компьютерных элементов сложной системы управления в процессе модернизации аппаратного и программного обеспечения.

Научная новизна. В процессе проведения исследований получены новые научные результаты:

— усовершенствована аналитическая модель качества вычислительной системы (далее — ВС), которая, в отличие от существующих, учитывает аппаратно-независимые ошибки, их качество и количество в исполняемом коде;

— разработан алгоритм поиска ошибок в исполняемом коде, который основан на существующих методах обратного проектирования, что позволяет обнаружить ошибки в готовом образце системы без технической документации и исходных кодов ПО;

— разработана методика оценки влияния ошибки в ПО, которая, в отличие от существующих, использует аппарат нечёткой логики, что позволяет решить проблему отсутствия статистических данных при переносе ПО на ААП;

— разработан метод оценки качества КЭ СУ, который, в отличие от существующих, использует анализ исполняемого кода для того, чтобы осуществить поиск ошибок, принять решение о возможной модификации ПО и установить контроль над функциональными компонентами системы.

Теоретическая значимость работы заключается в разработке моделей и методов, являющихся дополнением в методологии оценки базовых характеристик качества (функциональная пригодность, надёжность,

производительность) КЭ СУ с учётом полного или частичного отсутствия исходных кодов.

Практическая значимость работы состоит в том, что её результаты в условиях отсутствия технической информации о КЭ СУ (например, техническая документация, исходные коды и др.) позволяют рассчитать многокритериальную оценку характеристик качества на основе готового образца системы. Такой подход позволяет проводить оценивание качества систем, которые получены в результате переноса ПО на ААП.

Практическая значимость и новизна подтверждаются тем, что на основе предложенного метода оценки влияния ошибки на систему разработан комплект программ, защищённых авторским свидетельством.

Внедрение результатов диссертационной работы. Полученные результаты использованы (Приложение А) в учебном процессе Института кибербезопасности и цифровых технологий на кафедре «Интеллектуальные системы информационной безопасности» (структурного подразделения МИРЭА — Российского технологического университета (РТУ МИРЭА)), а также в научной и практической деятельности акционерного общества «АСТ» (АО «АСТ»).

Методы исследования. Решение поставленных в работе задач выполнено с использованием методов модельно-ориентированного системного инжиниринга, теории сетей Петри, теории массового обслуживания, теории вероятностей, теории нечёткой логии, теории логико-вероятностного исчисления. Для исследования эффективности разработанных методов и алгоритмов проводились экспериментальная проверка на встроенном программном обеспечении сетевых устройств и имитационное моделирование на ЭВМ.

Положения, выносимые на защиту:

1. Аналитическая модель качества компьютерных элементов системы управления, представляющая математическую зависимость оценки качества от параметров исполняемого кода.

2. Методика оценки влияния ошибки в программном обеспечении на характеристики качества компьютерных элементов системы управления,

использующая аппарат нечёткой логики для учёта разнородных метрических характеристик воспроизведения ошибки.

3. Метод оценки качества компьютерных элементов системы управления, основанный на анализе моделей исполняемого кода и алгоритме поиска программных ошибок.

Степень достоверности и апробация результатов. Основные положения диссертационной работы докладывались и обсуждались на следующих научных и научно-практических конференциях:

1. Научно-практическая конференция «Цифровые технологии и их приложения». Уфа, 2021 год.

2. V международная научная конференция «Приоритетные направления инновационной деятельности в промышленности». Казань, 2021 год.

3. Всероссийская научно-практическая конференция «Актуальные тренды цифровой трансформации промышленных предприятий». Казань, 2022 год.

4. Международная научная конференция «Актуальные проблемы прикладной информатики в образовании, экономике, государственном и муниципальном управлении». Барнаул, 2022 год.

5. XLVIII международная научно-практическая конференция «Российская наука в современном мире». Москва, 2022 год.

6. VII научно-техническая конференция «Математическое моделирование, инженерные расчёты и программное обеспечение для решения задач ВКО». Москва, 2022 год.

7. I национальная научно-практическая конференция «Кибербезопасность: технические и правовые аспекты защиты информации». Москва, 2023 год.

8. VIII научно-техническая конференция «Математическое моделирование, инженерные расчёты и программное обеспечение для решения задач ВКО». Москва, 2023 год.

9. III национальная научно-практическая конференция «Кибербезопасность: технические и правовые аспекты защиты информации». Москва, 2024 год.

Основные результаты диссертации опубликованы в 15 научных работах, из них: 4 опубликованы в рецензируемых научных изданиях, из перечня ВАК Минобрнауки России, 1 свидетельство государственной регистрации программы для ЭВМ, 10 — в других изданиях и материалах конференций. 2 статьи опубликованы в соавторстве.

Соответствие паспорту научной специальности 2.3.1 «Системный анализ, управление и обработка информации, статистика» по следующим направлениям:

1. Пункт 3. Разработка критериев и моделей описания и оценки эффективности решения задач системного анализа, оптимизации, управления, принятия решений, обработки информации и искусственного интеллекта.

2. Пункт 11. Методы и алгоритмы прогнозирования и оценки эффективности, качества, надёжности функционирования сложных систем управления и их элементов.

Личный вклад автора. Все основные результаты диссертации получены автором самостоятельно. Личный вклад диссертанта в работы, выполненные в соавторстве, состоит в разработке методов и алгоритмов поиска аппаратно-зависимых ошибок в исполняемом коде. В работе предлагается использовать усовершенствованные модели и методы оценки качества КЭ СУ при модернизации сложных систем управления. Экспериментальная проверка полученных результатов диссертации проведена автором самостоятельно.

Автор благодарит научного руководителя д. т. н. Новикова В. А. за помощь и поддержку в исследованиях, а также при подготовке диссертации. Автор выражает благодарность заведующему аспирантурой к. э. н. Брусову К. В. за оказываемое содействие, поддержку и методическую помощь при написании диссертации.

ГЛАВА 1. АНАЛИЗ МОДЕЛЕЙ И МЕТОДОВ ОЦЕНКИ КАЧЕСТВА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

1.1 Модели качества вычислительной системы

В процессах модернизации сложных технических систем необходимо оценивать качество составных элементов, в том числе вычислительных систем. Понятие «качество вычислительной системы» определяется как степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет таким образом оценить достоинства [6]. Под качеством понимается весь объём признаков и характеристик продукции или услуги, который относится к их способности удовлетворять установленным или предполагаемым потребностям [7].

Полнота, достоверность и точность оценки качества задаётся требованиями заинтересованных сторон. Качество создаваемой или существующей сложной технической системы зависит от оценки и прогнозирования качества создаваемых или поставляемых технологий, в том числе иностранного производства.

Существующие модели качества программных систем учитывают множество параметров или показателей качества (ПК), состояний оцениваемого продукта, а также ограничения, с которыми могут столкнуться все заинтересованные стороны в процессе оценивания. В научной литературе и в руководящих документах предлагаются различные методы оценки ПК, состояний и ограничений. Вместе с тем, существующие модели качества не позволяют адекватно рассчитывать многокритериальные оценки модернизируемых вычислительных систем, так как объектом их применения не являются аппаратные и программные компоненты совместно [10].

Большое количество общих свойств и характеристик в разнородных программно-аппаратных комплексах способствует созданию структурированной системы оценки качества. Такие системы разнородных характеристик называют

моделями качества вычислительных систем. В научной литературе разработано и описано множество моделей качества как программного обеспечения, так и программно-аппаратных систем. Нормативные документы [6, 7] по оценке качества информационных систем обобщают и расширяют модели Боэма, МакКола [11], FURPS [12], Дроми [13], Гилба, GQM+ [14].

Согласно руководящим документам ГОСТ Р ИСО/МЭК 25010-2015, модели качества делят на два класса — модели при использовании и внутренние модели системы.

Модель качества при использовании применяется в случаях оценки человеко-машинных систем в конкретных условиях. Такая модель содержит пять характеристик качества, таких как:

— эффективность;

— производительность;

— удовлетворённость;

— свобода от риска;

— покрытие контекста.

Внутренняя модель качества продукта характеризует вычислительную систему или программное обеспечение по статическим и динамическим свойствам. К характеристикам модели относятся:

— функциональная пригодность;

— уровень производительности;

— совместимость;

— удобство использования;

— надёжность;

— защищённость;

— сопровождаемость;

— переносимость.

Характеристики качества рассчитываются на основе ПК, которые определяются функциями, а входными параметрами этих функций являются

элементы показателей качества (ЭПК). ЭПК получают с помощью расчётных, измерительных, экспертных методов.

В ГОСТ 28195-89 [6] и ГОСТ Р ИСО/МЭК 25010-2015 [7] представлены обобщённая модель и методы расчёта качества ВС. Условно модель можно разделить на внутренние и внешние характеристики, которые определяются внутренними и внешними метриками.

Внутренние метрики характеризуют саму систему, а внешние — реакцию на взаимодействие с окружением. Разделение характеристик на внутренние и внешние позволяет грубо определить методы оценки ПК, а также — какие источники данных следует использовать. В [7] представлены два класса методов оценки: расчётный и экспертный. Важно отметить, что развитие оценки качества программно-аппаратных систем направлено на сокращение экспертных подходов с заменой на расчётные методы. Так как основным источником данных для внутренних метрик является система и техническая информация о ней, то её параметризация по количественным метрикам позволяет построить внутреннюю модель качества с использованием максимального количества расчётных методов. Такая модель в большей степени объективна и уточняется не за счёт экспертного знания, а за счёт знания о системе. Определим модель качества по внутренним характеристикам (Таблица 1).

Таблица 1 – Внутренние характеристики качества ВС и их показатели

Характеристики	Показатели
Функциональная пригодность	Полнота
	Корректность
	Целесообразность
Уровень производительности	Производительность
	Ресурсоёмкость
Надёжность	Завершённость
	Готовность
	Отказоустойчивость
	Восстанавливаемость
Сопровождаемость	Модульность
	Тестируемость
Переносимость	Адаптируемость
	Устанавливаемость

Такая обобщённая модель позволяет оценивать целевую систему как самостоятельно, так и в составе более общей системы по оценке качества. Стоит

отметить, что представленная модель не учитывает аспекты информационной безопасности, так как при оценивании безопасности используются собственные модели [15, 16] и полученные результаты могут быть интегрированы в общую оценку системы. Модель качества на основе внутренних характеристик имеет проблему, связанную с отсутствием данных о системе, что ведёт к решению задачи поиска необходимой информации в системе и о системе.

Представленная на Рисунке 1 модель основана на работах [1, 2, 3, 7] и адаптирована для расчётов оценки качества ВС.

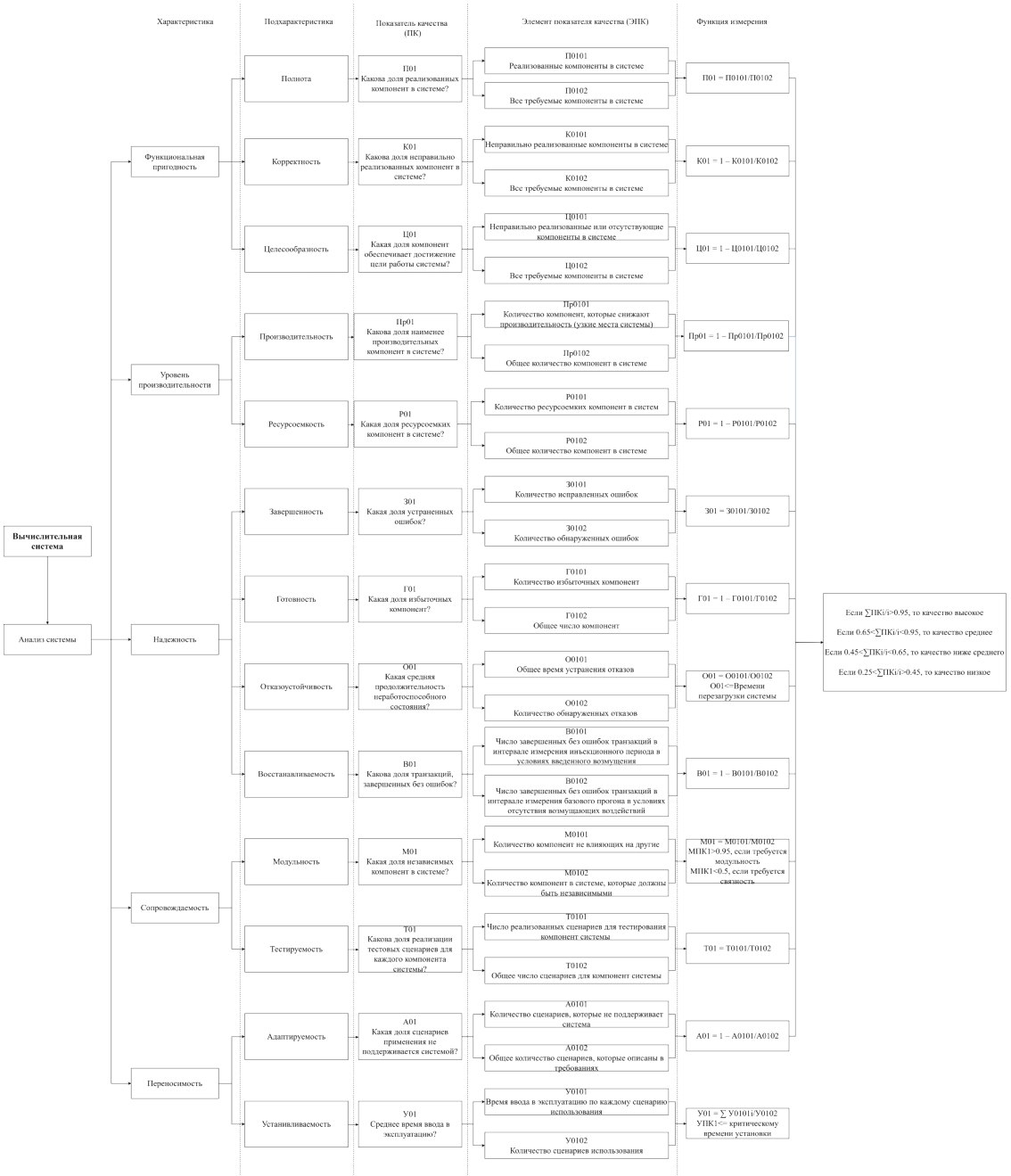


Рисунок 1 — Модель качества информационной системы

Модель включает 14 обобщённых показателей качества, каждый из которых отвечает на поставленный вопрос к структуре и функционированию системы. Ответом на каждый вопрос эксперт выбирает количественное значение из принятой системы оценки [17].

Понятие «компоненты системы» позволяет обобщить часть ПК, которые имеют общий смысл, но разные методы оценки.

На Рисунке 2 представлена архитектурная модель встраиваемой вычислительной системы.

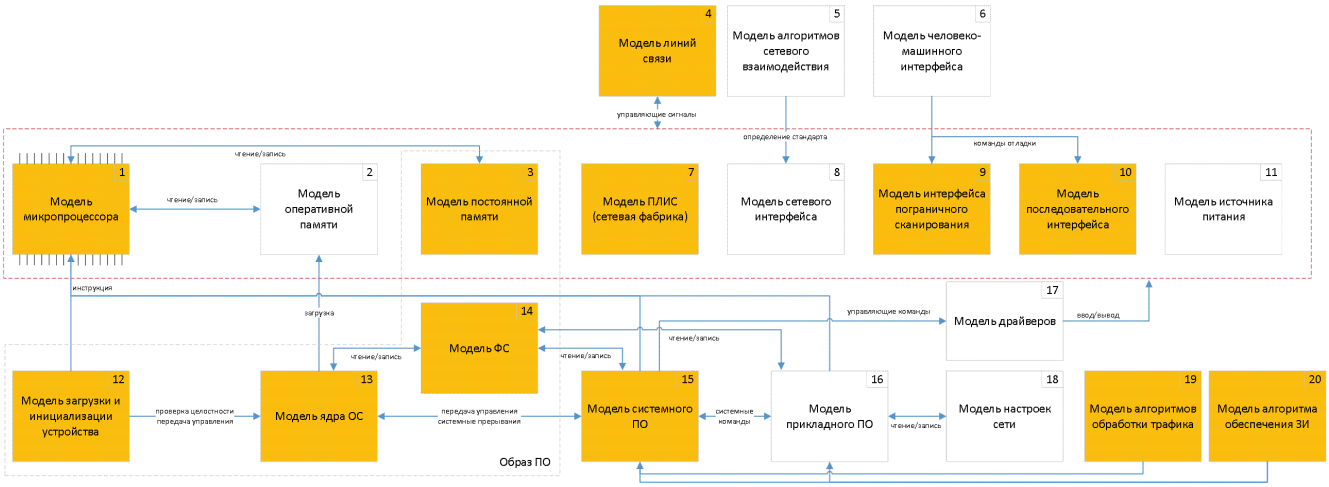


Рисунок 2 — Архитектурная модель встраиваемой ВС

Компоненты ВС делят на аппаратные и программные. Аппаратные, в свою очередь, – на базовые и обслуживающие. В базовые аппаратные компоненты входят процессоры, чипы памяти, чипы связи. В обслуживающие — конденсаторы, резисторы, транзисторы и т. д. [18]. Программные компоненты включают как общие понятия («операционные системы», «сервисы», «операционные подсистемы»), так и частные решения (программы, программные функции, программные модули, программные процедуры). Декомпозиция компонент системы позволяет декомпозировать обобщённые ПК и выделить ЭПК. ЭПК — это количественная метрика, которая соответствует логике поставленного вопроса. На Рисунке 3 представлен пример расчёта ПК для характеристики «производительность».

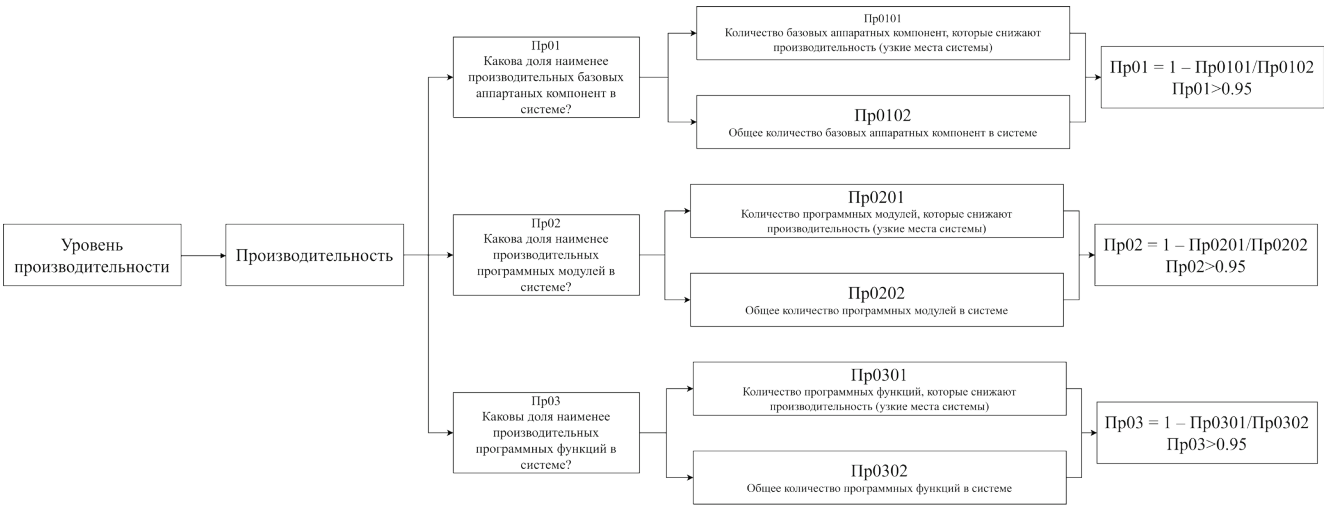


Рисунок 3 — Пример расчёта производительности ВС

Точность расчёта по представленной модели качества отвечает на вопрос «Какова доля показателей качества, которые удалось рассчитать?» (1):

$$Q = \frac{K}{S}, \tag{1}$$

где K — число рассчитанных ПК, а S — общее число всех ПК модели.

Ещё одним показателем точности произведённых расчётов может служить оценка погрешности при измерении показателей качества.

Точность расчёта зависит от исходных данных и методов измерения. Источники данных для расчёта ПК определены в [8] и включают следующие элементы:

— техническая документация системы;

— исходные тексты программного обеспечения;

— образец системы;

— руководства по эксплуатации;

— описание системы;

— требования потребителя системы;

— список обнаруженных отказов.

В работах [4, 5] определена аналитическая модель качества, которая несколько отличается содержанием, но структурно идентична тем моделям, что представлены в [6] и [7].

Эталонная модель качества ВС, которая описана в работе [4], может быть уточнена за счёт детализации ПК и применения систем массового обслуживания для расчёта производительности [19, 20].

Для унификации следует заменить понятие «атрибут качества» на «ПК» и добавить понятие «ЭПК». Тогда модель качества ВС записывается в виде

$$M_Q = \{Q, A, M, E, W\}, \quad (2)$$

где $Q = \{q_1, q_2, \dots, q_i\}\ i = 1, \dots, I$, — множество характеристик качества;

$A = \{a_1, a_2, \dots, a_j\}\ j = 1, \dots, J$, — множество ПК, каждый a_j отражает свойство характеристики качества q_i;

$M = \{m_1, m_2, \dots, m_k\}\ k = 1, \dots, K$, — множество метрик для нормализации каждого ПК a_j;

$E = \{e_1, e_2, \dots, e_n\}\ n = 1, \dots, N$, — множество ЭПК ПК a_j;

$W = \{w_1, w_2, \dots, w_l\}\ l = 1, \dots, L$, — множество весовых коэффициентов для множества М.

В работе Лаврищевой Е. М. [1] выбирается шесть характеристик качества:

— функциональность;

— надёжность;

— удобство применения;

— эффективность;

— сопровождаемость;

— переносимость.

Липаев В. В. [5] также выделяет шесть характеристик с единственным отличием в том, что он использует понятие «мобильность» вместо «переносимость».

Понятие «характеристика качества» объединяет множество ПК, что требует детализации и формализации. Оценка характеристик «функциональная пригодность», «надёжность», «производительность» и «защищённость» обладает полнотой при оценке качества ВС. Такие характеристики, как удобство применения, сопровождаемость, переносимость, вторичны в случаях, когда система не удовлетворяет требованиям базовых характеристик. Тогда выделим те

характеристики, требования к которым должны быть строгими для всех программно-аппаратных систем. К ним относятся:

q_1 — функциональная пригодность;

q_2 — надёжность;

q_3 — производительность;

q_4 — защищённость.

Существующие методики оценки ВС разделяют понятия качества и безопасности. Поэтому характеристика «защищённость» относится к общему понятию «безопасность информации» и, таким образом, её выделяют не как характеристику качества, а как характеристику безопасности.

Количественная оценка отдельных характеристик качества рассчитывается как сумма всех нормализованных ПК, умноженных на весовой коэффициент

$$q_i = \sum_{j=1}^{N} a_{ij} m_{ij} w_{ij}\,, \tag{3}$$

где N — количество ПК в q_i -ой характеристики

В свою очередь интегральная оценка качества ВС есть сумма всех характеристик качества.

$$Q = \sum_{i=1}^{3} q_i \tag{4}$$

1.2 Методы оценки качества программно-аппаратных систем

Методы оценки качества направлены на поиск, анализ, учёт и интерпретацию отказов в исследуемой системе. В теории надёжности технических систем важным критерием является количество и интенсивность отказов [21]. Отказы обусловлены внешними и внутренними факторами.

К внешним факторам относятся действия пользователей. Внутренними факторами являются дефекты в системе. Дефекты в вычислительной системе прежде всего связаны со случайными или преднамеренными ошибками в программном коде и аппаратных компонентах. Надёжность системы характеризуется такой цепочкой понятий, как «ошибка —дефект — отказ — системный отказ — сбой — системный сбой». Поэтому поиск ошибок является основой в методах оценки надёжности.

В то же время ошибки влияют не только на показатели надёжности, но и на другие ПК. Без определения количества и качества существующих ошибок в системе при расчёте функциональной пригодности нельзя рассчитать показатели функциональной полноты, корректности и точности, так как они зависят от таких ЭПК, как реализованная и корректная функция. Функция считается реализованной и корректной, если количество ошибок в ней не превышает границы допустимых значений и последствия от каждой не вызывают системные ошибки.

Ошибки в вычислительной системе также влияют на показатели производительности. При сокращении каналов обработки данных из-за ошибок сокращается пропускная способность системы и увеличивается время отклика, причём сама система может функционировать без сбоев. А в системах, где присутствуют ошибки работы с памятью («утечка памяти», дефекты в чипах памяти), зачастую увеличивают её расход в сравнении с теми же функциями без ошибок, таким образом сокращая доступный ресурс системы.

В зависимости от способов воздействия на систему выделяют измерительные методы (допускают вмешательство в систему) и регистрационные методы (пассивный сбор и анализ доступных данных).

1.2.1 Измерительные методы оценки качества

Измерительные методы основаны на получении информации о свойствах и характеристиках ВС с использованием инструментальных средств. Например, такие методы применяются для определения объёма программного кода (число строк исходного текста, число процедур, функций и т. д.) [6]. Они зависят от возможностей инструментирования системы.

Программный код ВС является основным источником угроз качеству. Большое количество логически сложных связей неизбежно связано с ошибками, которые снижают качество системы в целом. Выявление разного рода ошибок в ПО сопровождается параметризацией программного кода. Выделяются и измеряются следующие метрики: количество строк кода, количество функций, число выходов из программы и т. д. В [7] описаны наиболее значимые метрики программно-аппаратных систем. Специалисты выбирают метрики, которые доступны для измерения существующими методами и инструментами.

В [1] предлагается использовать связный набор метрик, который даёт целостное представление о качестве программного кода. Выделяют 4 группы метрик:

— сложность программного кода;

— связность программного кода;

— структурированность программного кода;

— объёмные характеристики программного кода.

Каждая группа включает набор известных метрик, например, сложность определяется совокупностью таких мер, как цикломатическое число Мак-Кейба [22], интервальная мера Майерса [23], мера Холстеда [24], а для объединения всех метрик используется мера Кокола [25].

Существующие инструменты разработки ПО автоматизируют процесс измерения программного кода. Анализ возможностей инструментов проведен в работе [26], где автор выделил следующие метрики:

— цикломатическая сложность;

— глубина наследования;

— сцепление классов;

— количество строк кода;

— количество вызовов функций;

— количество арифметических операций.

Представленные метрики измеряются инструментами Code Metrics (VisualStudio), MonoDevelop, VivaShowMetrics (PVS-Studio), SourceMonitor, а для отображения метрик на графе используют IBM Rational ClearCase. Пакет программ Understand от компании Scientifics Toolworks Inc. измеряет 104 метрики исходного кода, реализована поддержка API, доступна возможность для разработки собственных плагинов, отображение в виде UML-диаграмм [27].

В работе Диасамидзе С. В. [28] применяется метрический анализ с целью выявления недекларированных возможностей (НДВ). С помощью канонических схем Янова моделируется программное обеспечение в виде управляющего графа Г(В, Д), где $B = \{b_i\}$ – множество вершин (линейные участки программы), Д – множество связей по управлению. Схемы Янова позволяют привести логическую модель исполняемого кода к продуктивной структуре, которая подвергается метрическому анализу.

Автор выделяет 14 метрик и объединяет их в метрический базис. Базис делится на две части: метрики структуры программы и метрики информационной сложности. После чего ранжирует и определяет как наиболее предпочтительные следующие метрики: длина текста по Холстеду, метрика Джилба, точки пересечения, метрика интеллектуального содержания. Эти метрики сложности показывают высокий уровень корреляции искажений в программе с общим количеством дефектов. В то же время утверждается, что преднамеренные ошибки не влияют на метрики сложности исходной модели. Предлагается устранить в модели дефекты, которые присутствуют в программе, тем самым увеличивая чувствительность метрик.

Преимущества методов измерения программного кода в процессе оценки заключаются в следующем:

— простота измерения;

— высокая точность измерения;

— возможность автоматизации измерения практически любой метрики;

— интеграция со средой разработки;

— возможность измерения на всех этапах жизненного цикла (далее — ЖЦ) продукта.

Недостатком является низкая связность большого числа количественных показателей, которые в одних случаях могут указывать на проблемы с качеством, а в других не влияют на него. В качестве решения выступают систематизация и объединение метрик в группы.

Другая проблема заключается в том, что при отсутствии исходных кодов методы измерения не адаптированы для бинарного или промежуточного кодов. В следствии этого ПК, основанные на метриках программного кода, имеют низкую точность. Если учитывать, что статические методы поиска ошибок выдают около 20% ложных срабатываний [29], то точность ПК, рассчитанных на основе этих параметров, будет столь же низкой. Однако измерение программного кода — обязательная часть в программе оценки ВС, а методы измерения кода позволяют определить большое число ЭПК.

Наряду со статическими методами измерения программного кода используют динамические методы. Сущность динамических методов состоит в том, что процедуры измерения происходят в процессе выполнения программы [30]. Динамический анализ делится на два вида: ручной и автоматизированный. К инструментам ручного динамического анализа относятся среда разработки, программные отладчики, аппаратные отладчики. Автоматизированный анализ всегда связан с процессами инструментации программного кода. Под инструментацией программного кода понимается его модификация с целью установления процедур-перехватчиков, датчиков, мониторов для проведения

инструментальных измерений [31]. Обобщённая схема инструментирования представлена на Рисунке 4.

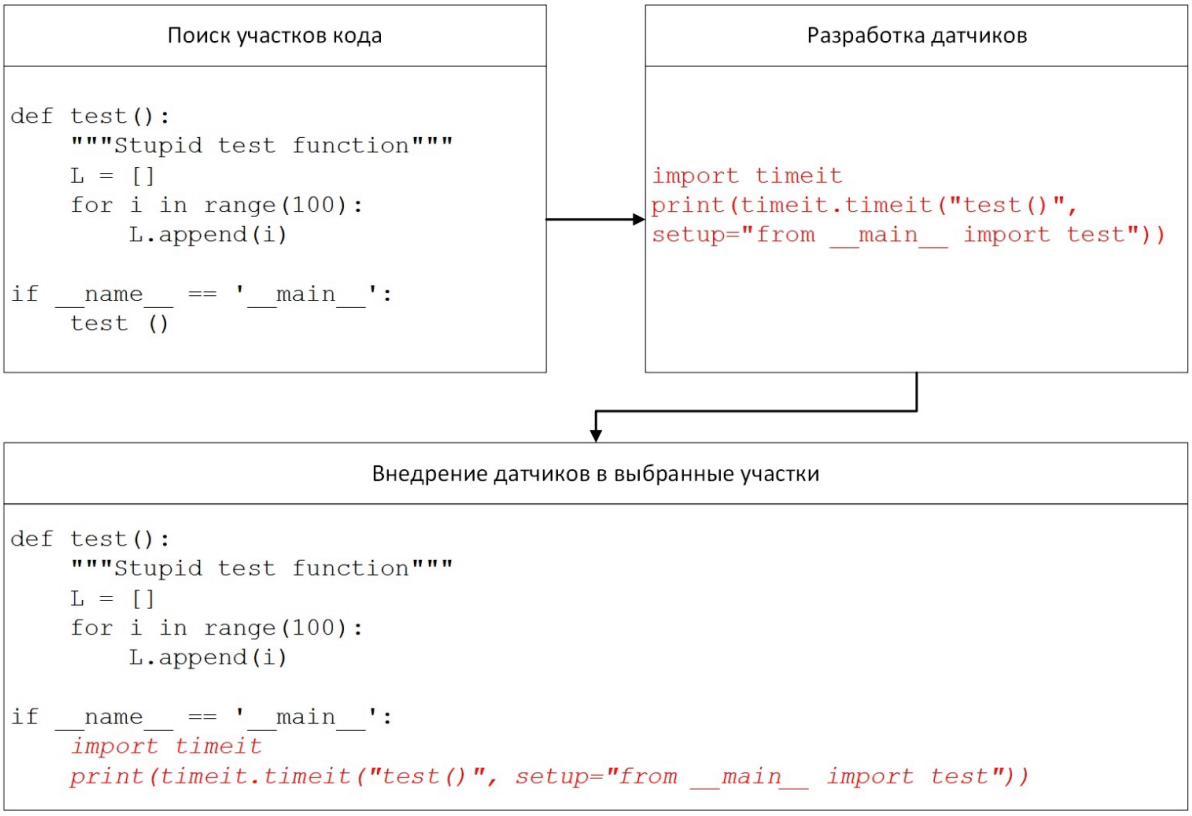


Рисунок 4 — Схема инструментирования программных компонент

Выделяют три типа инструментирования: модификация исходного кода программы, модификация исполняемого кода программы, модификация бинарного кода программы [32]. Подход к инструментированию определяется состоянием кода. Так, модификация исходного кода программы производится до компиляции, модификация исполняемого кода — в процессе выполнения, а модификация бинарного кода — между компиляцией и выполнением. Каждый метод инструментирования имеет особенности, которые способны влиять на точность конечного показателя качества. Поэтому целесообразно применять их совместно, чтобы компенсировать недостатки друг друга. Но в процессе оценки качества возникают ситуации, когда недостаточно исходных данных для использования одного или всех методов сразу. К таким проблемам относятся отсутствие исходных кодов, отсутствие доступа к бинарному коду, отсутствие возможности модифицировать исполняемый код. Подобные сложности способны сильно

ограничить процесс оценивания, что в итоге не позволит достичь требуемого уровня точности.

Аппаратная часть вычислительной системы состоит из базовых блоков: корпус, печатная плата, процессор (-ы), модули памяти, периферийное оборудование, источник питания, система охлаждения и другие компоненты. Каждый блок детализируется до требуемого уровня. Уровень детализации определяется в том числе и на этапе проведения измерений. В процессе оценки качества проводятся массогабаритные измерения, измерения тока и напряжения, мощности, измерение шумов и помех [33], измерение тепловых величин. Инструментирование аппаратных компонент является неотъемлемой частью методов измерения. В отличие от измерения программного обеспечения, средства измерения аппаратных компонент строго соответствуют спецификациям [34].

1.2.2 Регистрационные методы анализа ВС на основе тестирования образца системы

Регистрационный метод основан на получении информации во время испытаний или функционирования системы, когда регистрируются и подсчитываются определённые события, например, число сбоев и отказов, время передачи управления другим подсистемам, время начала и окончания работы [35, 36].

Функциональное тестирование — один из самых распространённых регистрационных методов, который позволяет оценить соответствие программно-аппаратной системы функциональным требованиям [37]. Суть его состоит в том, что оценщик разрабатывает тестовые программы, тестовые наборы данных или тестовые сценарии, использование которых показывает, как функционирует система в различных ситуациях. Все ответы регистрируются и сохраняются для дальнейшей обработки.

Выделяют ручное и автоматическое тестирование. В первом случае человек разрабатывает тесты, во втором — тесты генерируются автоматически на основе подобранного алгоритма под целевую систему. Ручное тестирование функций целесообразно на этапах разработки и малоэффективно в процессе оценки качества, так как увеличивает время на оценку. Куда более эффективно математическое тестирование. За счёт применения специализированных инструментов тестирования сокращается время на проверку сценариев, увеличивается объём покрытого кода. Проблема данного метода состоит в том, что сложно учитывать множество состояний системы, в которых она может функционировать. Ещё одна проблема связана с тем, что в процессе тестирования система находится в состоянии, отличном от рабочего режима. Так, например, система может определять, что её поведение отлично от рабочего режима и переходить в особый режим функционирования, напрямую связанный с обработкой тестовых запросов. В таком случае система успешно пройдёт тестовые испытания за счет недекларированного функционала [38].

Различают три уровня тестирования программно-аппаратных систем: модульное тестирование, интеграционное тестирование, системное тестирование [39]. Модульное тестирование подразумевает исполнение тестовых сценариев для отдельных компонент системы, а остальные компоненты должны не участвовать в обработке. Для этого в процессе разработки создаются тестовые сценарии для большинства функций или методов. Такой способ позволяет эффективно выявлять ошибки на всех этапах разработки. С другой стороны, с ростом сложности реализованного алгоритма в функции или методе количество тестов резко растёт, поэтому количество исполняемых тестовых сценариев ограничено имеющимися ресурсами. Модульное тестирование применяется как для тестирования программного обеспечения, так и для аппаратных компонент. В процессах ЖЦ ВС такой вид тестирования широко применяется в процессах проектирования и разработки.

Инструменты и среды разработки ПО интегрируют возможности по проведению тестирования. Компания Microsoft предлагает рекомендации

по модульному тестированию для .NET Core, .NET Standard [40]. Компания выделяет следующие преимущества модульного тестирования:

— сокращение времени на выполнение функционального тестирования;

— защита от регрессионных ошибок;

— документация в виде тестов;

— уменьшение связности кода.

Представленные выше преимущества справедливы, если тестирование производит разработчик; в случаях, когда тестирование применяется в процессе оценки качества, даже сокращение времени становится не столь очевидным. Прежде чем проводить модульное тестирование готовой системы, необходимо произвести её декомпозицию по тестируемым модулям, что накладывает дополнительные ограничения.

Цель интеграционного тестирования состоит в поиске ошибок при взаимодействии различных модулей и подсистем. Принципы интеграционного тестирования не отличаются от модульного, отличие состоит в объекте тестирования. В работе [39] применительно к интеграционному тестированию описаны методы нисходящего и восходящего тестирования.

Нисходящее тестирование — это процесс тестирования программно-аппаратной системы от главного модуля к вызываемым. Отдельно тестируется только главный модуль. Дальше тестируется совокупность модулей по мере их вызова в системе. Альтернативный метод — это восходящее тестирование. Отдельно тестируется только модуль, который вызывается последним. Затем в тестировании участвует вызывающий модуль, и так один за одним, вверх по цепочке, тестируется вся система. Существуют и другие методы интеграционного тестирования. Метод большого скачка, когда проводится модульное тестирование, а затем тестируются все модули сразу. Метод «сэндвича», когда одновременно запускают нисходящее и восходящее тестирование; тестирование происходит до тех пор, пока тестируемые уровни не встретятся.

Системное тестирование — это функциональное тестирование вычислительной системы в момент сопряжения всех компонент. В процессе

системного тестирования не допускается введение ограничений за счёт внедрения заглушек в систему. В научной литературе встречается понятие «системное тестирование» с другим смыслом, когда понятие «системный подход в тестировании» подменяет понятие «системное тестирование» [41]. Следует строго разделять эти понятия, поэтому под системным тестированием в диссертации понимается тестирование системы.

Системное тестирование отличается меньшей степенью детализации объекта тестирования. Решая задачу проверки аппаратных компонент, часто применяют имитационное моделирование на основе конечных автоматов. В процессе моделирования малых систем с небольшим количеством состояний метод модульного тестирования позволяет находить ошибки. В то же время с ростом числа состояний конечного автомата при моделировании сложных систем происходит комбинаторный взрыв [42]. Уменьшение детализации позволяет снизить число состояний конечного автомата и успешно применять метод системного тестирования.

Методы функционального тестирования являются определяющими регистрационными методами оценки качества ВС. На основе результатов тестирования рассчитывается большинство характеристик качества. В процессе оценивания ВС возникает ряд сложностей с проведением тестирования и получением точных результатов. Как правило, функциональное тестирование ВС при модернизации СУ проводится методом «чёрного» или «серого ящика». Методы «чёрного ящика» заключаются в анализе системы без исходных кодов и технической спецификации на компоненты. Тестирование методом «серого ящика» подразумевает частичное наличие исходных кодов (например, библиотеки с открытым исходным кодом) и технической информации о системе. Из этого следует, что применение методов модульного и интеграционного тестирования ограничено. В то же время оценщик не в состоянии адекватно оценить долю тестового покрытия ВС [43].

Тестирование с использованием метрик тестового покрытия увеличивает точность тестового набора за счёт анализа исходного кода системы и оценки метрик покрытия. Выделяют следующие оценки тестового покрытия [44]:

— покрытие формальных утверждений (statement coverage): отношение доли выполненных утверждений в тестовом наборе ко всем формальным утверждениям о системе;

— покрытие ветвей (branch coverage): доля покрытых ветвей среди всех исполняемых ветвей системы;

— покрытие условий (condition coverage): доля покрытых условий, принимающих различные значения, во всём наборе условий в системе.

Наряду с оценками тестового покрытия стоит задача оптимизации тестовых наборов. Ограничение ресурсов при тестировании не позволяет генерировать сверхбольшие наборы тестовых сценариев. Оптимизация тестирования связана с методами комбинаторного покрытия [44], генетическими алгоритмами [45], нейронными сетями [46].

Нагрузочное тестирование состоит из следующих этапов: тестирование производительности, стрессовое тестирование, объёмное тестирование, тестирование надёжности [9].

Тестирование производительности проводится с целью установления возможности функционирования системы под нагрузкой. Регистрируются такие ЭПК, как пропускная способность канала связи, время на обработку данных, время задержки, количество свободных ресурсов (память, загруженность ЦП) и др.

Тестирование на восстановление позволяет оценить работоспособность системы в условиях большого количества некоренных запросов. Регистрируются ЭПК системы, связанные с характеристикой качества «восстанавливаемость». Подробное описание процесса проведения тестирования и оценка восстанавливаемости изложены в ГОСТ Р ИСО/МЭК 25045-2015 [7].

Тестирование надёжности позволяет убедиться в том, что система способна обработать ожидаемую нагрузку в течение длительного периода времени.

Регистрируются такие события, как утечка памяти, перезагрузка системы или сервисов и другие события, влияющие на стабильность работы.

Тестирование в полной мере применяется в процессах модернизации ВС в сложных СУ. Такие методы не требуют наличия исходных кодов и применяются в условиях ограниченной информации о системе. Но важно отметить, что с уменьшением неопределённости в знаниях о системе увеличивается точность тестовых наборов и методов тестирования. В теории надёжности используются деревья отказов для формального представления знаний о системе.

Дерево отказов — это упорядоченное графическое представление логико-вероятностной связи случайных событий (дефектов, отказов, ошибок), приводящих к наступлению нежелательных событий [47]. Анализ дерева отказов состоит в выявлении условий, факторов, событий и их комбинаций, которые приводят или могут привести к наступлению нежелательного события (полный или частичный отказ системы, снижение уровня безопасности и др.). Построение деревьев отказа сложных технических систем занимает ключевую роль в теории надёжности. Возникновение события в подсистеме может не влиять на работоспособность системы, но возникновение цепочки событий может вызывать сбой в отдельной подсистеме или во всей системе. Оценка вероятности возникновения событий и расчёт вероятности цепочки позволяют определить узкие места в системе и предпринять необходимые меры по предотвращению. Общая схема построения деревьев отказа показана на Рисунке 5.

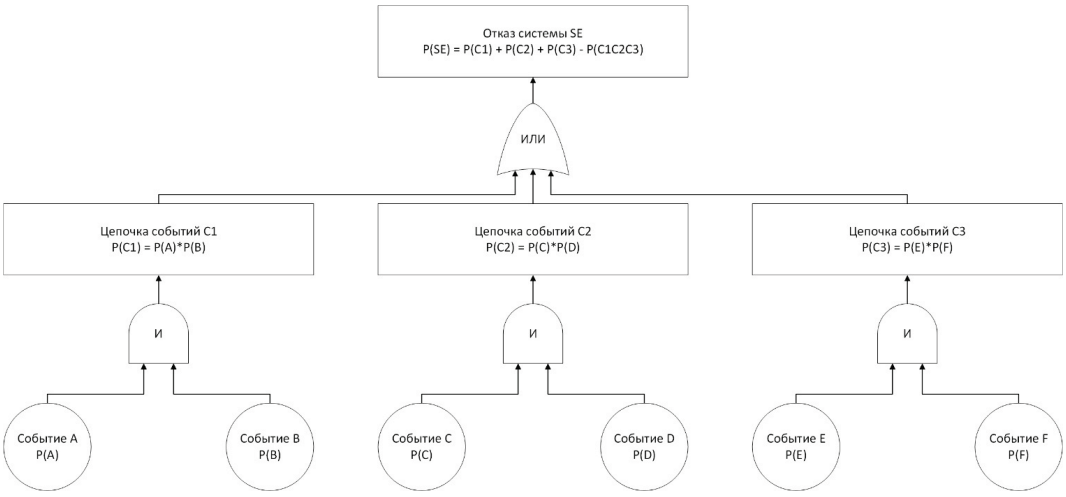


Рисунок 5 — Общее представление дерева отказов

Оценивается вероятность наступления базовых событий, после чего оценивается вероятность наступления главного события. Анализ логико-вероятностного представления нежелательных событий сложных систем позволяет сократить время на оценку слабых мест в системе.

ВС, безусловно, являются сложными техническими системами, где цепочка событий с определённой вероятностью ведёт к наступлению нежелательного события. Множество нежелательных событий определяет разработчик системы. В ситуациях, когда система оценивается как готовый образец, нежелательные события определяет оценщик. Вероятность наступления нежелательного события — это угроза качеству и безопасности системы.

Существуют официальные базы угроз для информационных систем. База ФСТЭК России [48] насчитывает 222 угрозы безопасности информации. Подобные банки данных затрагивают все аспекты, связанные с нарушением процессов хранения, обработки и передачи информации в вычислительной системе. Если рассматривать угрозы как преднамеренное воздействие на информацию, то это относится к области информационной безопасности и выходит за область данного исследования. Но если угрозы реализуются из-за технического состояния вычислительной системы, то их необходимо рассматривать как нежелательные события, что требует определения причин и оценки вероятности их наступления.

Рисунок 6 показывает, как возможно построить примитивное дерево отказов для угрозы УБИ.024 «Угроза изменения режимов работы аппаратных элементов компьютера» из базы данных ФСТЭК.

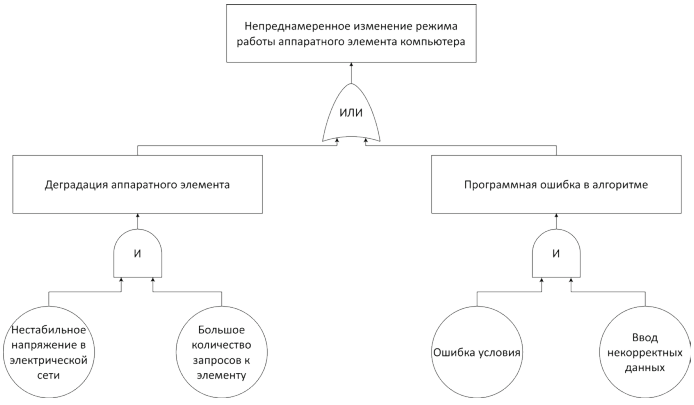


Рисунок 6 — Дерево отказа для угрозы УБИ.024 БД ФСТЭК

Построение деревьев отказа приобретаемой системы невозможно без системного анализа образца продукта. Оценка вероятностей наступления нежелательных событий в вычислительной системе не имеет смысла без восстановления связей, алгоритмов, протоколов взаимодействия программных и аппаратных компонент. Различные классы ошибок являются базовыми событиями, а их поиск и анализ найденных становятся важнейшей задачей.

Итак, обнаружение ошибок в компонентах системы занимает ключевую роль в оценке качества ВС. На разных этапах ЖЦ системы достаточность исходных данных отличается, но чем ближе система к готовому продукту, тем больше источников данных требуется для обнаружения ошибок в системе, так как состав методов зависит от доступности исходных данных (Рисунок 7).

Рисунок 7 — Методы оценки качества

В общем случае для оценки качества готовой вычислительной системы необходимы следующие исходные данные:

1. Готовый образец системы.
2. Техническая документация на систему.
3. Исходные коды программного обеспечения.
4. Технические требования к образцу системы.

В случае модернизации вычислительных систем в составе систем управления зачастую имеется только готовый образец и его техническое описание, что недостаточно для оценивания качества. В таких случаях исследователи ограничены методами тестирования и анализом открытых аналогов с целью получения количества ошибок, оставленных в системе.

1.3 Методы анализа исполняемого кода вычислительной системы

Обратное проектирование — процесс параметризации частных структурных и поведенческих моделей исследуемой системы с помощью анализа информации, добытой непосредственно из данной системы [49]. Обратное проектирование вычислительных систем включает различные методы, которые исследованы в отечественных работах по информационной безопасности Новикова В. А. [50, 51], Буйневича М.В. [52]; по разработке Лаврищевой Е. М. [53], Иванникова В. П. [54], Белеванцева А. А. [55]; по судебной экспертизе Тарасова Д. А. [56].

Зарубежные исследования по применению методов обратного проектирования ВС отличаются в зависимости от технологической развитости государств. В развитых странах такие работы связаны с обеспечением безопасности, определением заявленных характеристик, поиском незаконного заимствования собственных технологий, переносом технологий. Например, канадские учёные проводят исследования на предмет использования аналоговых технологий в цифровой среде [57]. Голландские учёные исследуют возможности восстановления документации на программное обеспечение методами обратного проектирования [58]. Технологически развивающиеся страны, такие как Китай [59] или Иран [60], помимо прочего применяют методы обратного проектирования для изучения существующих иностранных технологий.

Метод анализа исполняемого кода способен как самостоятельно решать задачу получения технической информации о системе и её параметризировать, так и применяться в дополнение к другим методам обратного проектирования [61, 62]. На Рисунке 8 представлена обобщённая схема анализа платформозависимого исполняемого кода.

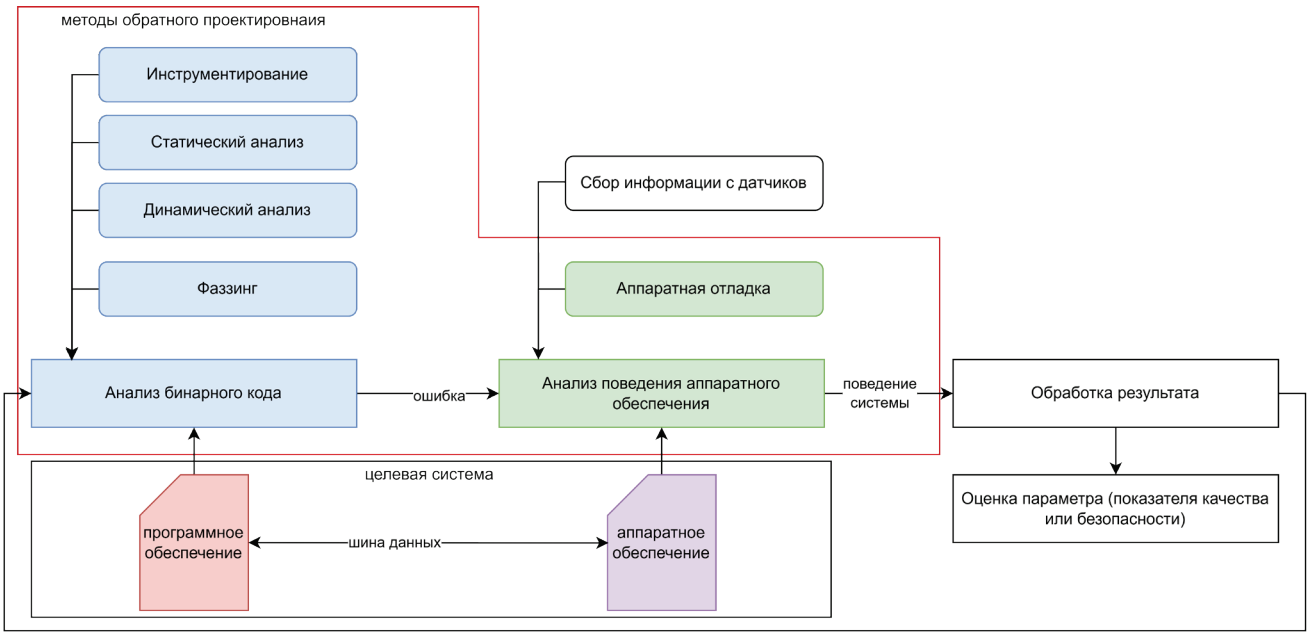


Рисунок 8 — Схема применения анализа исполняемого кода

На разных этапах ЖЦ системы достаточность исходных данных отличается, но чем ближе система к готовому продукту, тем больше источников данных требуется для обнаружения ошибок, так как состав методов зависит от доступности исходных данных. К методам обнаружения ошибок в вычислительной системе относятся:

— анализ технической документации;

— статический и динамический анализ исходного кода;

— статический и динамический анализ исполняемого кода;

— измерение параметров исходного кода;

— измерение параметров функционирования системы;

— аппаратная отладка системы;

— модульное, интеграционное и системное тестирование;

— регрессионное тестирование;

— нагрузочное тестирование;

— поблочное тестирование.

Таким образом, обнаружение ошибок в компонентах системы занимает ключевую роль в оценке качества вычислительных систем.

Поиск ошибок с помощью анализа исполняемого кода — устоявшийся процесс как в области разработки сложных программных систем, так и в области

кибербезопасности. Такой подход, с одной стороны, обусловлен отсутствием исходных кодов или их частичной утратой, а с другой — тем, что преобразования кода при компиляции под целевые платформы отличаются, и могут возникать дефекты из-за несовершенства в спецификациях языков программирования [63].

Существующие методы анализа исполняемого кода имеют единый подход, который основан на отображении бинарных данных в промежуточном представлении. Наиболее распространённым методом отображения является дизассемблирование бинарного (машинного) кода в ассемблер целевой архитектуры. Хотя при статическом анализе такой подход остаётся наиболее популярным, автоматизация этого анализа сопряжена со множеством проблем. Поэтому при развитии методов анализа исполняемого кода предлагаются другие промежуточные представления и методы их анализа.

При компиляции исходного кода некоторые компиляторы используют своё внутреннее представление. Так, например, LLVM (от англ. Low Level Virtual Machine) [64] представляет собой набор компиляторов, системы оптимизации, системы интерпретации исходных кодов на языке высокого уровня в исполняемый код. LLVM — это платформонезависимая виртуальная машина, которая использует специализированный ассемблер (байткод LLVM IR). Основным недостатком при анализе такого промежуточного представления является отсутствие информации о типах данных используемых переменных [65].

Растущая потребность в анализе исполняемого кода способствовала разработке специализированных промежуточных представлений. Vine [66] использует преобразование бинарных данных в промежуточный язык представления Vine IL. Другая платформа анализа исполняемого кода реализована в университете Карнеги — Меллона CMU BAP [67]. Она поддерживает аппаратные архитектуры x86_64, ARM, MIPS, PowerPC, механизмы разработки дополнительного функционала, а также взаимодействие с инструментами символьного выполнения. Также нельзя обойти стороной промежуточное представление, которое назвали языком обратного проектирования (от англ. Reverse Engineering Intermediate Language, REIL) [68]. Язык используется

в популярном инструменте обратного проектирования BinNavi, а его преимуществом является ограниченный набор инструкций — 17. Для сравнения, набор инструкций для PowerPC содержит более 1000 инструкций. Малое количество команд позволяет максимально упростить алгоритмы анализа кода REIL.

Анализ промежуточного представления исполняемого кода условно делится на статический и динамический. Динамический анализ делится на конкретное выполнение, символьное выполнение и абстрактную интерпретацию (Рисунок 9) [65].

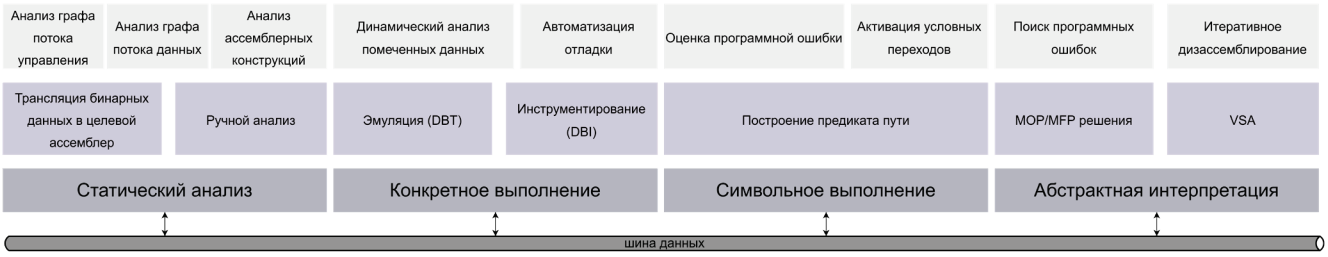


Рисунок 9 — Разновидности анализа исполняемого кода

На практике широко применяются инструменты, автоматизирующие анализ исполняемого кода. В рамках статического анализа к таким инструментам относятся Ida Pro, Ghidra, radare2, compose, objdump. При эмуляции на основе динамической бинарной трансляции (DBT) и динамической бинарной инструментации (DBI) используются инструменты QEMU, Valgrind, Pin, DynamoRIO. Символьное выполнение основано на реализации SMT-решателей в S2E, Mayhem, Sydr. Абстрактная интерпретация, с некоторым допущением, относится к статическому анализу, однако создание и итеративное применение передаточных функций является прообразом исполнения кода, что позволяет её выделить в отдельный вид анализа. Задачи анализа абстрактных синтаксических деревьев (от англ. Abstract syntax tree, AST) решают такие инструменты, как BinSide, BitBlaze, BAP.

Таким образом, имея готовый образец системы, исполняемые коды программных компонент, возможно восстановить алгоритмы функционирования и технические требования к образцу системы. Тем самым задача по оценке оставленных ошибок в вычислительной системе сводится к той, что решается

разработчиками на этапах проектирования, разработки и внедрения. Отсюда следует, что, используя методы анализа исполняемого кода, возможно рассчитывать ПК и дать многокритериальную оценку качества вычислительной системе в условиях ограниченного набора исходных данных.

1.4 Анализ точности методов оценки качества вычислительных систем в процессе переноса ПО на ААП

Точность измерений качества в области сложных технических систем является относительным параметром, который не может быть зафиксирован в силу того, что опорные измерения изменяются во времени. В науке методы измерений используют такие критерии, как правильность и прецизионность [69]. Правильность в методах измерений использует понятия истинного или опорного значения.

В рамках оценки точности метода измерения предполагается, что каждый результат измерений, y, представляет собой сумму трех составляющих

$$y = m + B + e, \tag{5}$$

где m — математическое ожидание, B — лабораторная составляющая систематической погрешности в условиях повторяемости, e — случайная составляющая погрешности каждого результата измерений в условиях повторяемости.

Преобразуем исходную модель (5) путём замены математического ожидания m на

$$m = \mu + \delta, \tag{6}$$

где μ — принятое опорное значение математического ожидания измеряемой величины; δ — систематическая погрешность метода измерений.

Подставив (6) в (5), получим следующую запись исходной модели:

$$y = \mu + \delta + B + e, \tag{7}$$

Составляющая e модели (7) является случайной погрешностью, которая не зависит от метода измерений и лаборатории, в которой проводились измерения. Если принять случайную погрешность как постоянную повторяющуюся величину, то модель (7) принимает следующий вид:

$$M(y) = \mu + \delta + B, \tag{8}$$

где $M(y)$ — математическое ожидание измеряемой величины y.

Количество выявленных ошибок в реализации вычислительной системы зависит от методов поиска ошибок, а также от различных условий, в которых находится лаборатория. Поэтому получаемые оценки качества вычислительной системы зависят как от систематической погрешности метода оценки δ, так и от лабораторных условий, которые характеризуются погрешностью B. В рамках исследования введём общий показатель систематической погрешности оценки качества вычислительной системы Δ

$$\Delta = \delta + B \tag{9}$$

Тогда на основе модели (8) систематическую погрешность оценки качества возможно вычислить по следующей формуле:

$$\Delta = |M(y) - \mu| \tag{10}$$

Расчёты точности методов измерений физических характеристик объекта используют истинные значения измерений на эталонном образце; в ситуациях, когда эталон не существует, используются принятые опорные значения, которые получены как:

1) Установленные значения, которые основаны на научных принципах.
2) Установленные значения, базирующиеся на экспериментальных работах.
3) Математическое ожидание измеряемой характеристики.

В методах оценки качества ВС используются принятые опорные значения, которые являются средними значениями заданной совокупности результатов оценки. В ситуациях, когда используются ААП для функционирования, существующего ПО, опорными значениями являются те результаты оценки, которые получены в условиях контроля всех этапов разработки системы.

1.4.1 Проблемы выявления ошибок в программном коде при переносе ПО на ААП

Оценка качества прежде всего направлена на анализ системы с целью поиска преднамеренных или случайных ошибок, оставленных в программном коде. Согласно (9), систематическая погрешность оценки Δ зависит от систематической погрешности метода оценки (отсутствия возможности применить методы поиска ошибок) и лабораторных условий (неправильное и/или неполное применение методов поиска ошибок), что опосредованно является вероятностью невыявленных ошибок.

Пусть количество ошибок определяется как $e = e_s + e_h$, где e_s — это количество аппаратно-независимых ошибок, а e_h — это количество аппаратно-зависимых ошибок. Найденные ошибки обозначим следующим образом:

$e^s = e_s^s + e_h^s$ — количество ошибок, найденных в ходе статического анализа исходных кодов;

$e^d = e_s^d + e_h^d$ — количество ошибок, найденных в ходе динамического анализа исходных кодов;

$e^f = e_s^f + e_h^f$ — количество ошибок, найденных в ходе формальной верификации исходных кодов;

$e^h = e_s^h + e_h^h$ — количество ошибок, найденных в ходе аппаратной отладки системы;

$e^t = e_s^t + e_h^t$ — количество ошибок, найденных в ходе тестирования системы.

Отсюда следует, что остаточное количество ошибок определяется формулами:

$$e_s' = e_s - \left(e_s^s + e_s^d + e_s^f + e_s^h + e_s^t\right) \quad (11)$$

$$e_h' = e_h - \left(e_h^s + e_h^d + e_h^f + e_h^h + e_h^t\right) \quad (12)$$

$$e' = e_s' + e_h' \quad (13)$$

Обозначим систему S с остаточным количеством ошибок e' как $S(e')$.

Одна из основных задач при переносе ПО на ААП состоит в том, чтобы изменённая система удовлетворяла требованиям качества по всем характеристикам; иными словами, качество системы должно быть не ниже, чем у исходного состояния системы. Рассмотрим возможные исходы переноса ПО на ААП (Рисунок 10).

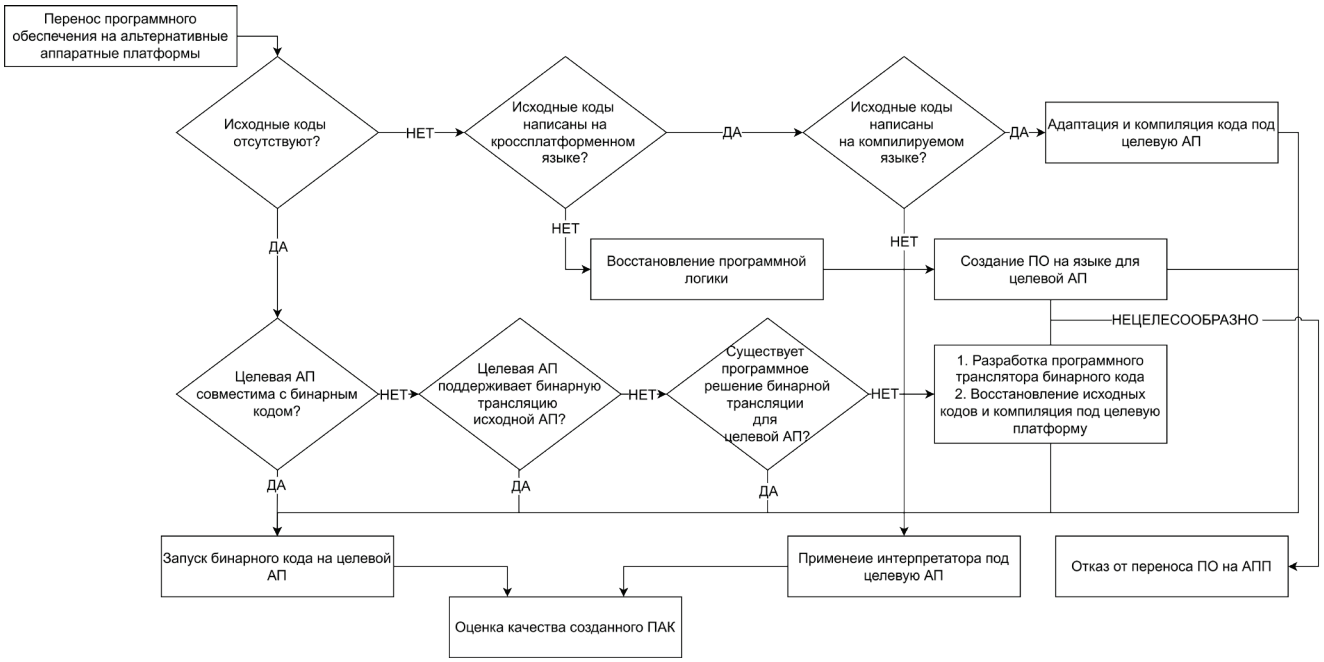


Рисунок 10 — Схема переноса ПО на ААП

Обозначим $S(e')$ как исходное состояние системы с остаточным количеством ошибок e', тогда получаем следующие состояния системы при переносе ПО на ААП:

$S_{11}(e'_{11})$ — исходные коды отсутствуют, и целевая аппаратная платформа совместима с бинарным кодом;

$S_{12}(e'_{12})$ — исходные коды отсутствуют, и целевая аппаратная платформа поддерживает бинарную трансляцию исходной АП;

$S_{13}(e'_{13})$ — исходные коды отсутствуют, и существует программное решение бинарной трансляции для целевой АП;

$S_{14}(e'_{14})$ — исходные коды отсутствуют, и существует возможность разработки программного транслятора бинарного кода;

$S_{15}(e'_{15})$ — исходные коды отсутствуют, и существует возможность их восстановления;

$S_{16}(e'_{16})$ — исходные коды отсутствуют, и отсутствует возможность переноса ПО на ААП;

$S_{21}(e'_{21})$ — ПО использует интерпретатор, который доступен для ААП;

$S_{22}(e'_{22})$ — исходные коды в наличии, но требуются их адаптация и компиляция под ААП;

$S_{23}(e'_{23})$ — исходный код ПО написан на специализированном языке АП и требует портирования под целевой язык ААП;

$S_{24}(e'_{24})$ — исходный код ПО написан на специализированном языке АП, и отсутствует возможность переноса на ААП.

Состояния $S_{16}(e'_{16})$ и $S_{24}(e'_{24})$ не требуют оценки качества системы, поэтому не учитываются в данном исследовании. Остаточное количество ошибок в состояниях S_{11}, S_{12}, S_{13}, S_{14} определяется как

$$e'_{11}, e'_{12}, e'_{13}, e'_{14} = e'_s + e^s_h + e^d_h + e^f_h + e^h_h + e^t_h, \quad (14)$$

причём в приведённых состояниях системы обнаружить ошибки e^s_h, e^d_h, e^f_h существующими методами не представляется возможным, так как отсутствуют исходные коды.

$S_{15}(e'_{15})$ и $S_{23}(e'_{23})$ содержат количество ошибок, сравнимое с количеством ошибок в исходной системе до проведения её анализа на ошибки, так как для того, чтобы привести систему в состояния S_{15} и S_{23}, необходимо создать новый исходный код.

Состояние $S_{22}(e'_{22})$ содержит остаточные аппаратно-независимые ошибки и все аппаратно-зависимые $e'_{22} = e'_s + e_h$, но так как исходные коды имеются в наличии, то существует возможность применить все существующие методы анализа системы.

И только в состоянии $S_{21}(e'_{21})$ точность оценки качества системы сохраняется, так как используется интерпретируемый язык, который работает одинаково на всех поддерживаемых аппаратных платформах.

Тогда вероятность обнаружения всех ошибок существующими методами анализа равна $P(e)$, а вероятность оставшихся ошибок — e': $P(e')$.

Введём следующие обозначения:

N — множество всех существующих методов обнаружения аппаратно-независимых ошибок;

$R = \{r | r \in N \text{ и } A(r)\}$ — множество применимых методов обнаружения аппаратно-независимых ошибок (правило $A(r)$) для конкретного состояния системы);

$M = \{m | m \in N \text{ и } m \notin R\}$ — множество неприменимых методов обнаружения аппаратно-независимых ошибок для конкретного состояния системы;

K — множество всех существующих методов обнаружения аппаратно-зависимых ошибок;

$Z = \{z | z \in K \text{ и } \Phi(z)\}$ — множество применимых методов обнаружения аппаратно-зависимых ошибок (правило $\Phi(z)$) для конкретного состояния системы);

$C = \{c | c \in K \text{ и } x \notin Z\}$ — множество неприменимых методов обнаружения аппаратно-зависимых ошибок для конкретного состояния системы.

Вероятность выявления всех ошибок определим как сумму вероятностей обнаружения всех ошибок каждым методом:

$$P(e) = P(e_s^1) + P(e_s^2) + \cdots + P(e_s^n) + P(e_h^1) + P(e_h^2) + P(e_s^k) = \\ = \sum_{n=1}^{i} P(e_s^n) + \sum_{k=1}^{j} P\left(e_h^k\right), \quad (15)$$

где $n \in N$, k $\in K$, $P(e_s^n)$ — вероятность обнаружения аппаратно-независимых ошибок n-м методом, а $P\left(e_h^k\right)$ — вероятность обнаружения аппаратно-зависимых ошибок k-м методом.

Следовательно, вероятность оставшихся ошибок в ВС определяется как сумма вероятностей оставшихся ошибок от каждого метода:

$$P(e') = P(e_s'^1) + P(e_s'^2) + \cdots + P(e_s'^n) + P(e_h'^1) + P(e_h'^2) + \cdots + P(e_s'^k) = \\ = \sum_{n=1}^{i} P(e_s'^n) + \sum_{k=1}^{j} P\left(e_h'^k\right), \quad (16)$$

где $P(e_s'^n)$ — вероятность оставшихся аппаратно-независимых ошибок после

применения n-го метода, а $P\left(e_h'^k\right)$ — вероятность оставшихся аппаратно-зависимых ошибок после применения k-го метода

В тех случаях, когда метод поиска ошибок в ВС применить не удаётся, то вероятности оставшихся аппаратно-независимых ошибок определяются по формуле (17).

$$P(e_s') = \sum_{m=1}^{i} P(e_s^m) + \sum_{m=1}^{i} P(e_s'^m) + \sum_{r=1}^{i} P(e_s'^r), \quad (17)$$

где $P(e_s^m)$ — вероятность метода обнаружения аппаратно-независимых ошибок, который не удалось применить при анализе системы; $P(e_s'^m)$ — вероятность оставшихся ошибок в условиях применимости m-го метода; $P(e_s'^r)$ — вероятность оставшихся ошибок после метода, который удалось применить при анализе системы.

Аналогично (17) записывается формула (18), которая определяет вероятность оставшихся аппаратно-зависимых ошибок.

$$P(e_h') = \sum_{c=1}^{j} P(e_h^c) + \sum_{c=1}^{j} P(e_h'^c) + \sum_{z=1}^{j} P(e_h'^z), \quad (18)$$

где $P(e_h^c)$ — вероятность метода обнаружения аппаратно-зависимых ошибок, который не удалось применить при анализе системы; $P(e_h'^c)$ — вероятность оставшихся ошибок в условиях применимости c-го метода; $P(e_h'^z)$ — вероятность оставшихся ошибок после метода, который удалось применить при анализе системы.

Отсюда следует, что при применении всех существующих методов формулы (17) и (18) имеют вид:

$$P(e_s') = \sum_{m=1}^{i} P(e_s'^m) + \sum_{r=1}^{i} P(e_s'^r) \quad (19)$$

$$P(e_h') = \sum_{c=1}^{j} P(e_h'^c) + \sum_{z=1}^{j} P(e_h'^z) \quad (20)$$

Выразим (15) и (16) согласно формулам (17) и (18) и получим общие формулы оценки вероятностей обнаружения ошибок:

$$P(e) = \sum_{r=1}^{i} P(e_s^r) + \sum_{z=1}^{j} P(e_h^z) \tag{21}$$

$$P(e') = \sum_{m=1}^{i} P(e_s^m) + \sum_{m=1}^{i} P(e_s'^m) + \sum_{r=1}^{i} P(e_s'^r) + \\ + \sum_{c=1}^{j} P(e_h^c) + + \sum_{c=1}^{j} P(e_h'^c) + \sum_{z=1}^{j} P(e_h'^z) \tag{22}$$

Рассчитаем возможные вероятности обнаружения всех ошибок для исходной системы S при переносе её ПО на ААП. Зададим априорные вероятности $P(e) = \{0{,}9;\ 0{,}8;\ 0{,}7;\ 0{,}6\}$, которые соответствуют уровню точности в условиях применения $(n + k)$ методов анализа системы. Ограничим задачу следующими утверждениями:

— методы обнаружения аппаратно-независимых ошибок *равновероятные*;

— методы обнаружения аппаратно-зависимых ошибок *равновероятные*;

— методы обнаружения аппаратно-независимых и аппаратно-зависимых ошибок *не равновероятные*.

Приведём расчёты вероятности обнаружения ошибок для состояний системы $S_{11}, S_{12}, S_{13}, S_{14}$, учитывая, что в исходном состоянии
$P(e) = 0{,}9;\ P(e_s) = 0{,}45;\ P(e_h) = 0{,}45.$

$$S_{11}, S_{12}, S_{13}, S_{14}: \begin{cases} P(e) = 0{,}45 + 2 * 0{,}09 = 0{,}63 \\ P(e') = 0 + 0 + 5 * 0{,}01 + 3 * 0{,}09 + 3 * 0{,}01 + 2 * 0{,}01 = 0{,}37 \end{cases}$$

При заданных условиях вероятность обнаружения всех ошибок составляет 63%, что на 27% меньше исходной.

В Таблице 2 представлены расчёты, которые указывают на значительное снижение вероятности выявления ошибок в системе при переносе ПО на ААП.

Таблица 2 — Вероятность обнаружения ошибок в состояниях $S_{11}, S_{12}, S_{13}, S_{14}$

$P(e)$	$P(e_s)$	$P(e_h)$	$P(e)$ при $S_{11}, S_{12}, S_{13}, S_{14}$	Снижение обнаружения ошибок, %	Критичность снижения точности
0,9	0,9	0	0,9	0	Низкая
0,8	0,8	0	0,8	0	Низкая
0,7	0,7	0	0,7	0	
0,6	0,6	0	0,6	0	
0,9	0,8	0,1	0,84	6,67	
0,8	0,7	0,1	0,74	7,5	
0,7	0,6	0,1	0,64	8,57	
0,6	0,5	0,1	0,54	10	
0,9	0,7	0,2	0,78	13,33	
0,8	0,6	0,2	0,68	15	
0,7	0,5	0,2	0,58	17,14	
0,9	0,6	0,3	0,72	20	Средняя
0,6	0,4	0,2	0,48	20	
0,8	0,5	0,3	0,62	22,5	
0,7	0,4	0,3	0,52	25,71	
0,9	0,5	0,4	0,66	26,67	
0,8	0,4	0,4	0,56	30	
0,6	0,3	0,3	0,42	30	
0,9	0,4	0,5	0,6	33,33	
0,7	0,3	0,4	0,46	34,29	
0,8	0,3	0,5	0,5	37,5	
0,9	0,3	0,6	0,54	40	Высокая
0,6	0,2	0,4	0,36	40	
0,7	0,2	0,5	0,4	42,86	
0,8	0,2	0,6	0,44	45	
0,9	0,2	0,7	0,48	46,67	
0,6	0,1	0,5	0,3	50	
0,7	0,1	0,6	0,34	51,43	
0,8	0,1	0,7	0,38	52,5	
0,9	0,1	0,8	0,42	53,33	
0,9	0	0,9	0,36	60	Критичная
0,8	0	0,8	0,32	60	
0,7	0	0,7	0,28	60	
0,6	0	0,6	0,24	60	

На Рисунке 11 представлены графики вероятности обнаружения всех ошибок в состояниях $S_{11}, S_{12}, S_{13}, S_{14}$ в зависимости от вероятности обнаружения аппаратно-зависимых ошибок $P(e_h)$ при заданных исходных вероятностях $P(e) = 0{,}9$ (Рисунок 11а), $P(e) = 0{,}8$ (Рисунок 11б), $P(e) = 0{,}7$ (Рисунок 11в), $P(e) = 0{,}6$ (Рисунок 11г).

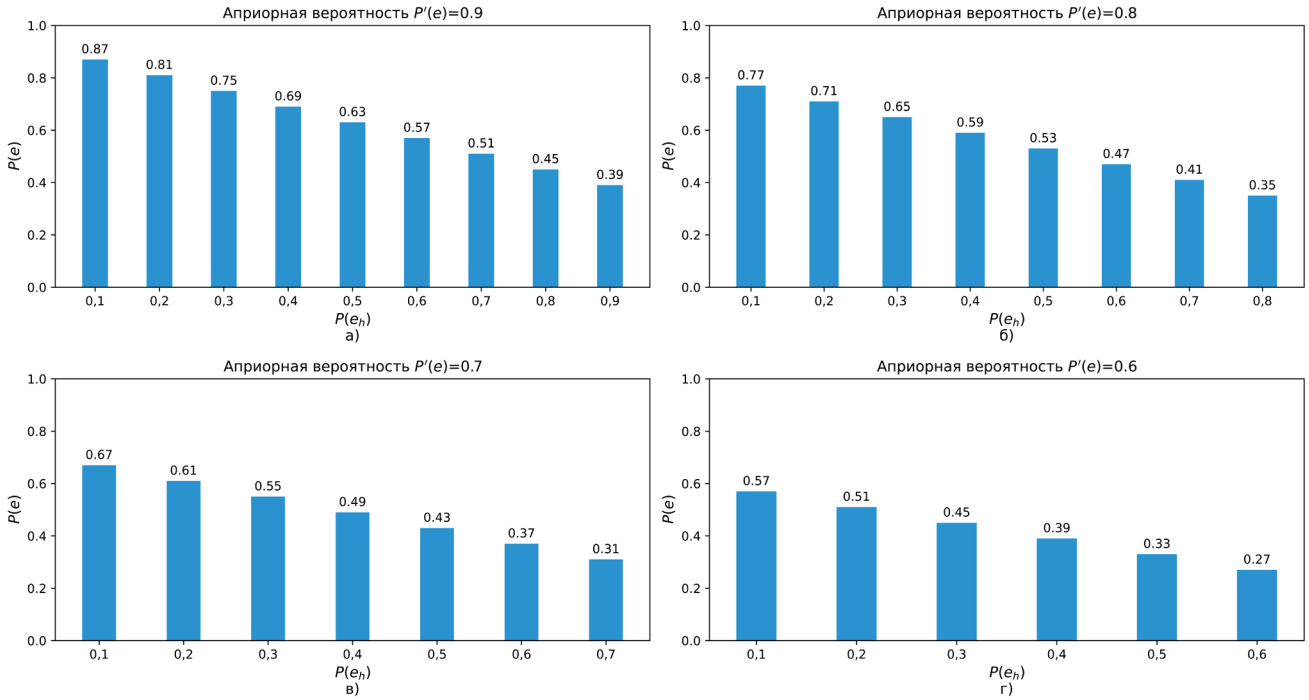


Рисунок 11 — Влияние аппаратно-зависимых ошибок на вероятность обнаружения всех ошибок в системе

Представленные расчёты отражают возможные ситуации, которые зависят от доступности методов поиска ошибок, методик их применения, а также от уровня компетенций специалистов. Реальная ситуация, которая сложилась в области разработки сложных программных систем под заявленную аппаратную платформу, соответствует расчётам, где априорная вероятность обнаружения всех ошибок $P(e)$ находится в интервале [0,8; 0,6] [70]. Для того чтобы определить вероятности обнаружения аппаратно-зависимых ошибок $P(e_h)$, проведён анализ отчётов об изменениях в программном обеспечении встраиваемых систем [71, 72]. На Рисунке 12 представлена круговая диаграмма распределения ошибок по типам.

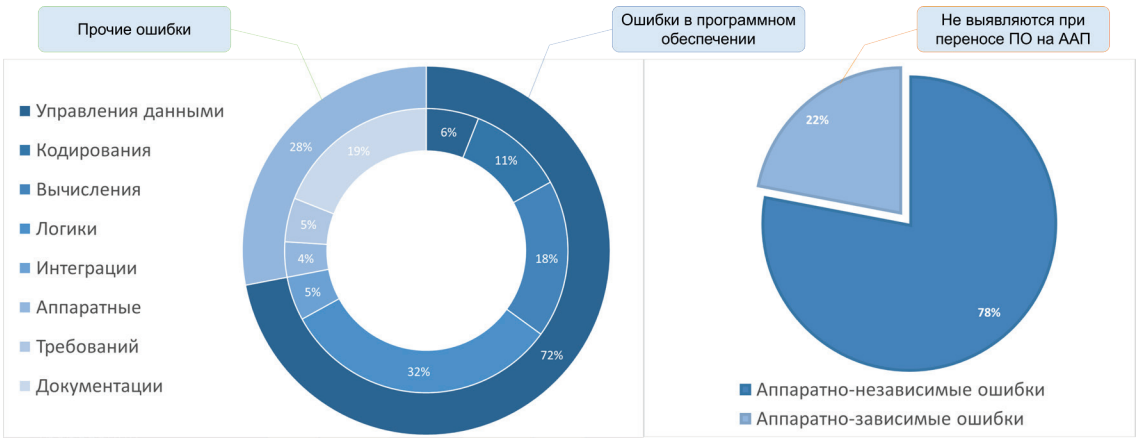


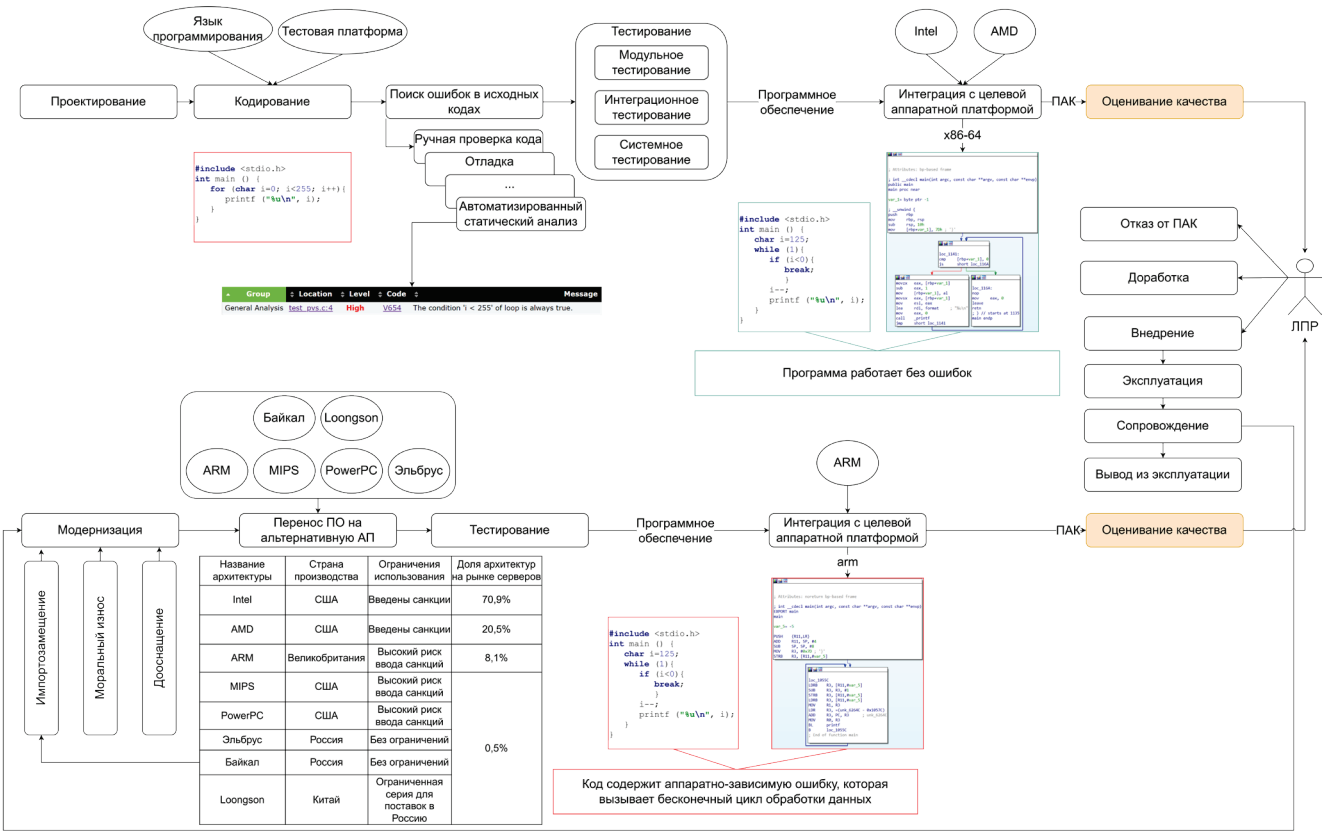
Рисунок 12 — Распределение ошибок во встраиваемых системах

Анализ данных об исправлениях в программном коде встраиваемых систем [73, 74] указывает на то, что через пять лет после выпуска оборудования более 22% всех обнаруженных ошибок составляют программные ошибки, связанные с аппаратным обеспечением. Это связано с тем, что используется единая кодовая база для поддерживаемых аппаратных платформ; иначе говоря, это ошибки, возникающие в момент переноса ПО на ААП. Отсюда следует, что истинная вероятность обнаружения аппаратно-зависимых ошибок $P(e_h)$ находится в интервале [0,8; 0,2]. Таким образом, практическая вероятность обнаружения всех ошибок в состояниях переноса ПО на ААП находится в интервале [0,68; 0,24], что в среднем на 24 % меньше исходной и соответствует теоретическим расчётам (27%).

1.4.2 Систематическая погрешность оценки качества в процессе модернизации компьютерных элементов системы управления

Программно-аппаратные комплексы, которые обеспечивают приём, передачу, обработку и хранение информации в сложной системе управления, являются компьютерными элементами системы управления (КЭ СУ). В отличие от пользовательских или серверных ВС, КЭ СУ обладают узкими функциональными возможностями, продолжительным временем эксплуатации; как следствие, к ним предъявляются повышенные требования по надёжности и производительности. Создание КЭ СУ предполагает контроль качества на каждом этапе ЖЦ.

На процесс модернизации сложных систем управления влияют факторы, которые ограничивают ресурс на создание нового ПО и/или АО, что вынуждает проводить синтез программно-аппаратных комплексов из уже имеющихся аппаратных компонент с последующим портированием программного кода на альтернативную платформу. На Рисунке 13 представлена обобщённая схема модернизации КЭ СУ с применением переноса ПО на ААП.



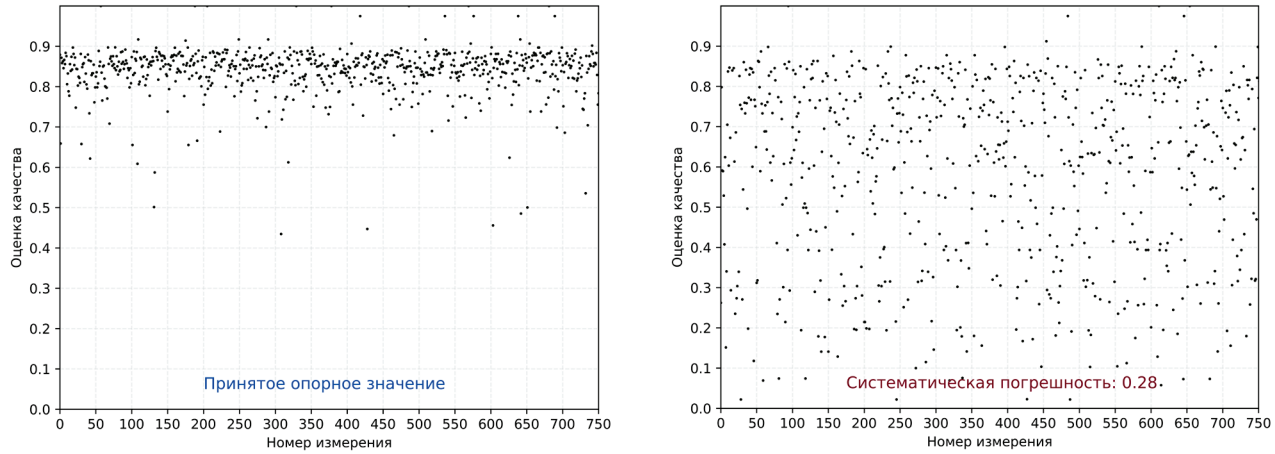
Название архитектуры	Страна производства	Ограничения использования	Доля архитектур на рынке серверов
Intel	США	Введены санкции	70,9%
AMD	США	Введены санкции	20,5%
ARM	Великобритания	Высокий риск ввода санкций	8,1%
MIPS	США	Высокий риск ввода санкций	0,5%
PowerPC	США	Высокий риск ввода санкций	
Эльбрус	Россия	Без ограничений	
Байкал	Россия	Без ограничений	
Loongson	Китай	Ограниченная серия для поставок в Россию	

Рисунок 13 — Схема модернизации КЭ СУ в условиях импортозамещения аппаратных компонент

На схеме цветом выделены процессы оценивания качества: после создания программно-аппаратного комплекса (далее — ПАК) по полному ЖЦ и после модернизации ПАК путём портирования программного кода. Это объясняется тем, что лицо, принимающее решение (ЛПР), должно обладать информацией о характеристиках КЭ СУ после внесённых изменений. Более того, точность оценки качества должна быть не ниже той, которая получена при создании ПАК. На практике получается так, что в силу возникающей систематической погрешности применяемых моделей и методов точность оценки качества значительно снижается.

Для того чтобы количественно оценить систематическую погрешность применяемых моделей и методов оценки качества при переносе ПО, в качестве опорных значений используем системы с открытым исходным кодом, который портируется под множество аппаратных платформ. При разработке КЭ СУ по полному ЖЦ применяются сложные аналитические модели оценки качества. Они включают не только базовые ЭПК, но и оценки ошибок с весовыми

коэффициентами [75]. Применение статических анализаторов на системах с открытым исходным кодом показало, что 21% всех ошибок имеют «высокий» (high) уровень критичности, 17% — «средний» (medium) и 62% — «низкий» (low). Модели оценки качества при переносе ПО на ААП не обладают такой информацией, так как учитывают только ошибки, вероятность обнаружения которых менее 68%. Используя эти данные в качестве исходных, проведены симуляции более 1000 возможных состояний КЭ СУ, и для каждой рассчитаны оценки качества по функциональной пригодности (Рисунок 14).



а) б)

Рисунок 14 — Расчёты оценки качества для исходной системы и при переносе ПО на ААП

Согласно формуле (10), систематическая погрешность оценки качества КЭ СУ составляет 28%, что не может удовлетворять требуемому уровню точности при принятии решений в процессе модернизации систем управления.

1.5 Выводы

— Проведён анализ существующих моделей качества информационных систем. В научной литературе представлены обобщённые модели качества. Описываются разобщённые показатели без привязки к конкретным моделям качества. В том числе, модели недостаточно показывают влияние характеристик

качества друг на друга. Следовательно, необходимо дополнить существующие модели качества конкретными показателями, свойственными при модернизации систем, и установить возможные зависимости между параметрами системы и показателями качества.

— Проведена декомпозиция методов оценки качества по выделенным характеристикам. Проанализированы возможности методов для оценки выделенных в модели показателей качества. Выявлены проблемы применения существующих методов оценки качества для неконтролируемых информационных систем. В частности, методы измерения программного обеспечения требуют исходные коды программ. В случае закрытых программно-аппаратных систем методы неприменимы.

— Существующие модели и методы способны оценить качество информационных технологий только в том случае, когда все показатели качества рассчитаны или рассчитаны риски, которые компенсируют неопределённость в конкретном показателе. В ином случае точность оценки не будет удовлетворять текущим требованиям.

ГЛАВА 2. МОДЕЛИРОВАНИЕ ОЦЕНКИ КАЧЕСТВА КОМПЬЮТЕРНЫХ ЭЛЕМЕНТОВ СИСТЕМЫ УПРАВЛЕНИЯ НА ОСНОВЕ БАЗОВЫХ ХАРАКТЕРИСТИК

Компьютерный элемент системы управления — это вычислительная система S с совокупностью аппаратных (H_i) и программных компонент (P_i), которые обладают целостностью и обеспечивают приём, передачу, обработку и хранение информации как от объекта управления, так и от других элементов сложной системы управления. Общая модель, учитывающая компоненты H_i и P_i системы S, характеризуется:

— уровнями иерархии моделирования;

— древовидными графами, где вершинами являются компоненты системы и компоненты компонент;

— наличием связей как внутри компонентов H_i и P_i, так и между ними;

— видами связей c_i;

— таблицами отношений сущностей по видам связей c_i.

С точки зрения анализа, определения свойств, поиска ошибок и, как следствие, оценивания качества КЭ СУ, целесообразно разделить ВС на компоненты и создать класс моделей под каждую характеристику качества. В системном инжиниринге для описания и представления технических систем применяется модельно-ориентированный подход (от англ. Model Based System Engineering, MBSE). Сбалансированный комплекс частных моделей образует метамодели, которые описывают сложные системы [76].

Метамоделирование ВС рассматривается по методу «сверху вниз». На Рисунке 15 представлена опорная стратификация моделей для оценивания качества ВС.

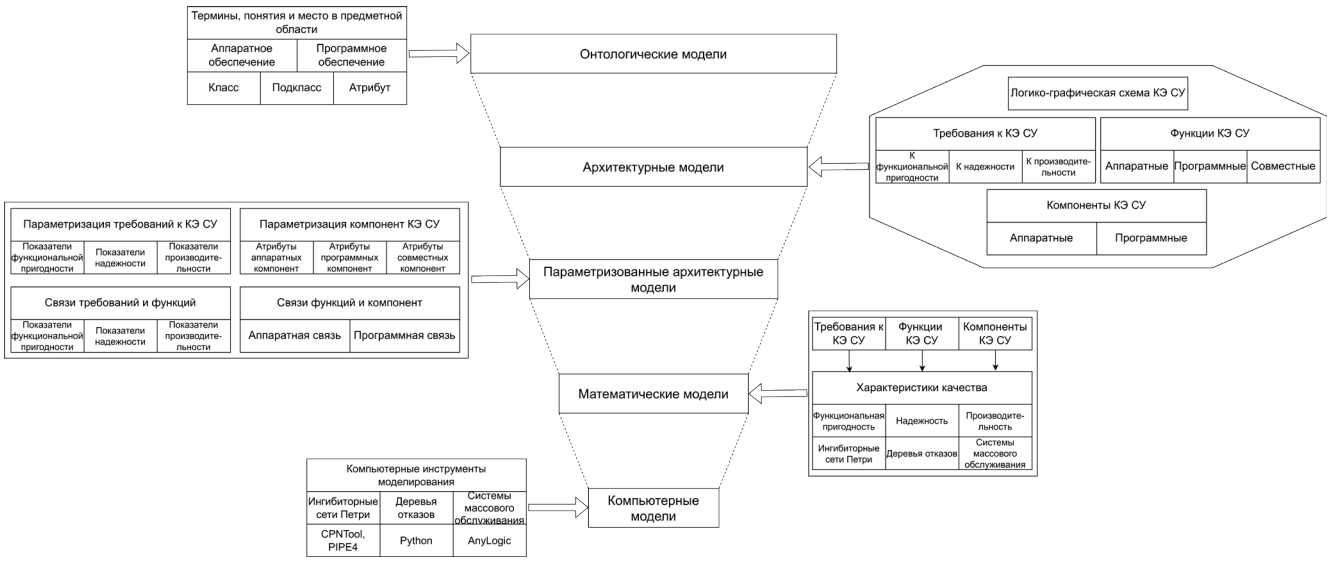


Рисунок 15 — Стратификация моделей качества ВС

Введение качественных и количественных параметров в модели позволяет создавать классификаторы и справочники данных. С одной стороны, качественные и количественные данные наполняют математические модели параметрами, а с другой — эти параметры являются элементами ПК (ЭПК — измеряемые величины) при оценке КЭ СУ.

2.1 Построение аналитической модели оценки качества КЭ СУ

2.1.1 Объединение разнородных показателей качества сложной программной системы в аналитическую модель

Первичное представление системы состоит из иерархической структуры системы (System Breakdown Structure, SBS), связей c_i и таблиц отношений $DSMk\ (SBS, SBS)$. На следующем шаге моделируются требования (Requirement Breakdown Structure, RBS) в соответствии с характеристиками качества, функции (Function Breakdown Structure, FBS) и компоненты (Product Breakdown Structure, PBS) системы, а также создаются таблицы отношений (Design Structure Models, DSM) с обозначением программных и аппаратных связей. Параметризация

моделей основана на определении метрик a_k сущностей E_i, которые представляются в виде количественных величин. На Рисунке 16 представлена иерархическая модель качества, в которой выделено 3 характеристики качества и 12 ПК [77].

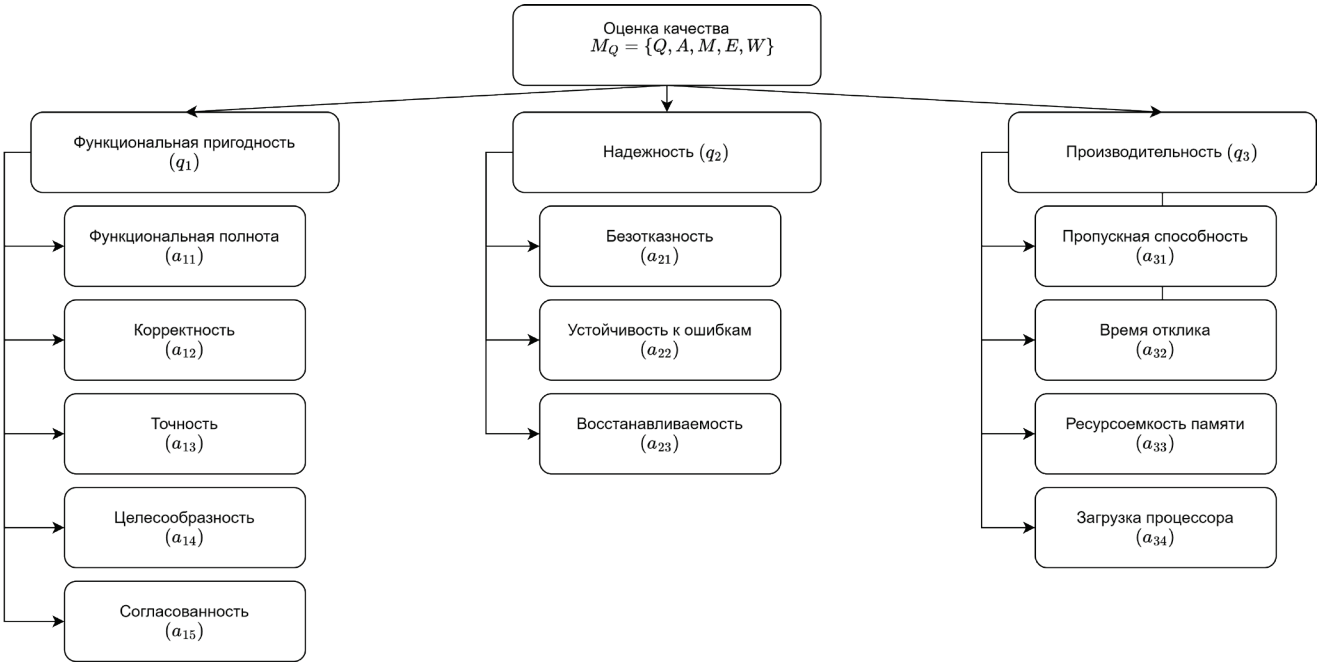


Рисунок 16 — Иерархическая модель оценки качества КЭ СУ

Пользуясь эталонной моделью качества вычислительной системы, составим модель, по которой оцениваются базовые характеристики.

Функциональная пригодность определяется *функциональной полнотой* (a_{11}) [78], *корректностью* (a_{12}) [79], *точностью* (a_{13}) [80], *целесообразностью* (a_{14}) [81], *согласованностью* (a_{15}) [82]: $q_1 = \{a_{11}, a_{12}, a_{13}, a_{14}\}$

Функциональная полнота — это отношение всех реализованных функций (F^c) в вычислительной системе к функциям, заданным в требованиях (F^m):

$$a_{11} = \frac{\sum_{i=1}^{N} F^c}{\sum_{j=1}^{K} F^m} \tag{23}$$

Корректность — это соответствие реализованных функций (F^c) заданным функциям в требованиях (F^m):

$$a_{12} = 1 - \frac{|F^m \backslash F^c|}{|F^m|} \tag{24}$$

Выделяют полную корректность, когда все реализованные функции соответствуют заданным в требованиях ($F^c = F^m$), и частичную корректность, когда $F^c \cap F^m \neq \emptyset$.

Точность — это свойство системы, которое отражает правильность полученных значений функциями на входном наборе данных. Точность оценивается как сумма разницы значений индикаторных функций, которые зависят от входных данных.

$$a_{13} = \sum_{i=1}^{|F_m|}((F_i^c(D_i) - F_i^m(D_i))/F_i^m(D_i))/|F^m|, \quad (25)$$

где D — это множество всех входных значений; Dr — это множество, на котором функция $F_i^m(D_i) = 1$; отсюда следует, что функция $F_i^m(D_i) = \begin{cases} 1, \text{если } D_i \in Dr \\ 0, \text{если } D_i \notin Dr \end{cases}$

Из данного определения следует, что если $D = Dr$ и функция $F_i^c(D_i)$ всегда принимает правильное значение, то показатель $a_{13} = 0$ и можно сделать вывод о том, что нет разницы между требуемыми значениями и получаемыми от системы. В таких случаях говорят, что система работает точно в соответствии с требованиями.

Целесообразность — это отношение компонент системы, обеспечивающих её функционирование (F^a), к общему количеству реализованных компонент в системе (F^c):

$$a_{14} = \frac{\sum_{i=1}^{N} F^a}{\sum_{j=1}^{K} F^c} \quad (26)$$

Согласованность — это отношение функций, реализованных в соответствии со стандартами (F^s), ко всем реализованным функциям (F^c):

$$a_{15} = \frac{\sum_{i=1}^{N} F^s}{\sum_{j=1}^{K} F^c} \quad (27)$$

Надёжность вычислительных систем определяется *безотказностью* (a_{21}) [83], *устойчивостью к ошибкам* (a_{22}) [84], *восстанавливаемостью* (a_{23}) [85].

Безотказность — это способность системы функционировать без отказов программных и аппаратных компонент. Безотказность оценивается вероятностью p безотказного выполнения компонента на случайно выбранном наборе входных

данных. Для вычислительных систем применяется такой показатель, как средняя наработка на отказ:

$$a_{21} = \frac{\sum_{i=1}^{m} t_i}{m}, \quad (28)$$

где t_i — интервал времени безотказной работы при наступлении i-го отказа, m — количество отказов.

Устойчивость к ошибкам — это способность отдельных компонент и системы в целом функционировать в нехарактерных условиях. Степень устойчивости системы оценивается как отношение различных типов отказов (N^*) к общему количеству отказов, возникающих в системе (N):

$$a_{22} = \frac{N^*}{N} \quad (29)$$

Восстанавливаемость — это способность системы восстанавливать функционирование после возникновения отказа. Восстанавливаемость оценивается с помощью измерения среднего времени, затраченного на восстановление:

$$a_{23} = \frac{\sum_{i=1}^{m} t_i^r}{m}, \quad (30)$$

где t_i^r — интервал времени, который необходим системе для восстановления после i-го отказа; m — количество отказов.

Модели систем массового обслуживания смешанного типа [86] позволяют описать вычислительную систему с целью расчёта показателей производительности.

Производительность вычислительных систем определяется *пропускной способностью* (a_{31}) [87], *временем отклика* (a_{32}) [88] и *ресурсоёмкостью* (a_{32}).

Пропускная способность — это отношение среднего числа обработанных заявок за единицу времени к среднему числу поданных заявок. Относительная пропускная способность системы — это вероятность того, что заявка в момент времени t не получит отказ.

$$a_{31}(t) = 1 - p_n(t), \quad (40)$$

где $p_n(t)$ — вероятность того, что заявка в момент времени t получит отказ.

Время отклика — это сумма среднего времени ожидания заявки ($T_{\text{ож}}$) и среднего времени обслуживания ($T_{\text{об}}$)

$$a_{32} = T_{\text{ож}} + T_{\text{об}} \quad (41)$$

Ресурсоёмкость — это количество используемых ресурсов при выполнении функции системы. Ресурсы компьютерной системы — это совокупность вычислительной мощности процессора (-ов) и используемой памяти. Ресурсоёмкость за единицу времени выражается как совокупность загруженности процессора и памяти

$$a_{33} = \{Cl(t), Vm(t)\}, \quad (42)$$

где $Cl(t)$ — это загруженность процессора за время выполнения функции, а $Vm(t)$ — степень используемой памяти от общего количества за время выполнения функции.

2.1.2 Онтологические и архитектурные модели КЭ СУ

Рассматривая КЭ СУ как объект оценивания качества, необходимо описать ограничения исследования:

— система является готовым образцом;

— производителем предоставляется техническая документация на готовый образец;

— отсутствует детальное описание программных и аппаратных компонент системы;

— полностью или частично отсутствуют исходные коды программного обеспечения.

Требуется создать опорную стратификацию КЭ СУ согласно схеме, представленной на Рисунке 16, с целью дальнейшей оценки характеристик качества системы.

Онтологическая модель КЭ СУ описывает термины, понятия и место в предметной области, где применяется оборудование [4]. Чтобы задать систему

понятий, прежде всего определяются основная задача и место компьютерных элементов в системе управления. К основным задачам КЭ СУ относятся:

— обеспечение функционирования канала связи между элементами СУ;

— приём, обработка и хранение информации о работе СУ;

— синхронизация сигналов между элементами СУ;

— контроль параметров процесса управления;

— цифровое регулирование в процессе управления.

В соответствии с правилами разработки онтологий проводится анализ КЭ СУ и выделяются основные понятия — классы (аппаратное обеспечение общего назначения (АО ОН), системное программное обеспечение (системное ПО), подсистема обработки и передачи данных), подклассы (микропроцессоры, контроллеры, чипы памяти) и атрибуты (разрядность, частота шины данных, напряжение). На Рисунке 17 предлагается обобщённая структурная схема элементов онтологии КЭ СУ.

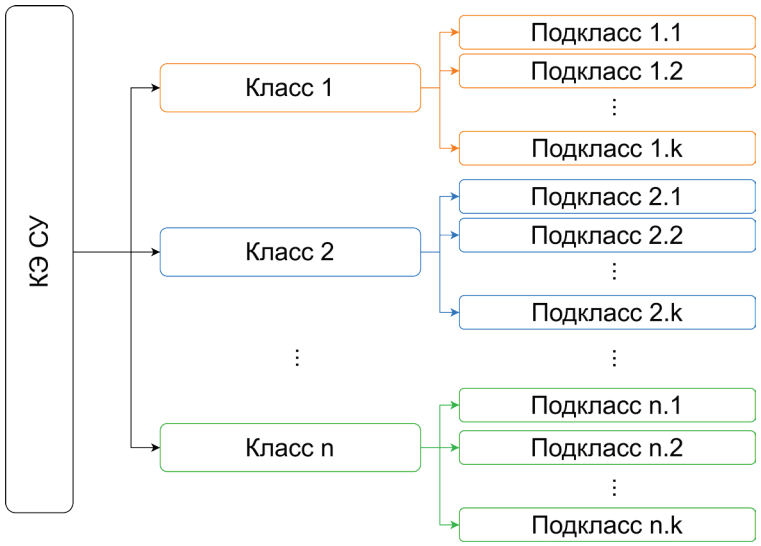


Рисунок 17 — Структура элементов онтологии КЭ СУ

На примере класса «Аппаратное обеспечение общего назначения» представим возможные подклассы и атрибуты (Таблица 3).

Таблица 3 — Онтология аппаратного обеспечения общего назначения КЭ СУ

Класс	Подкласс	Атрибут
Аппаратное обеспечение общего назначения	Микропроцессор	Архитектура
		Разрядность
		Тактовая частота
		Размер кэш-памяти

Продолжение таблицы 3

Класс	Подкласс	Атрибут
Аппаратное обеспечение общего назначения	Чипы памяти	ОЗУ
		ПЗУ
		Память микропроцессора
	Интерфейсы отладки	Интерфейсы отладки микропроцессора
		Интерфейсы отладки памяти
		Интерфейсы отладки периферийных устройств
		Визуальные интерфейсы (световые индикаторы)
	Источник питания	Схемы подключения
		Входное напряжение
		Выходное напряжение
		Мощность
		Количество выходов
		Температурные характеристики

Онтологические модели позволяют формализовать КЭ СУ в виде словарей терминов и логических взаимосвязей. Создание онтологии КЭ СУ позволяет перейти на следующий уровень абстракции — архитектурные модели.

Построение архитектурных моделей основано на создании различного рода схем, которые включают исследуемую часть моделируемого объекта с неотъемлемыми компонентами и связями. Архитектурные модели могут быть созданы как в виде простых схем, так и с помощью формальных языков моделирования (Unified Modeling Language (UML), Architecture View Model (AVM), Architecture Description Language (ADL)). На Рисунке 18 представлена схема КЭ СУ, обеспечивающего прием, передачу и обработку информации.

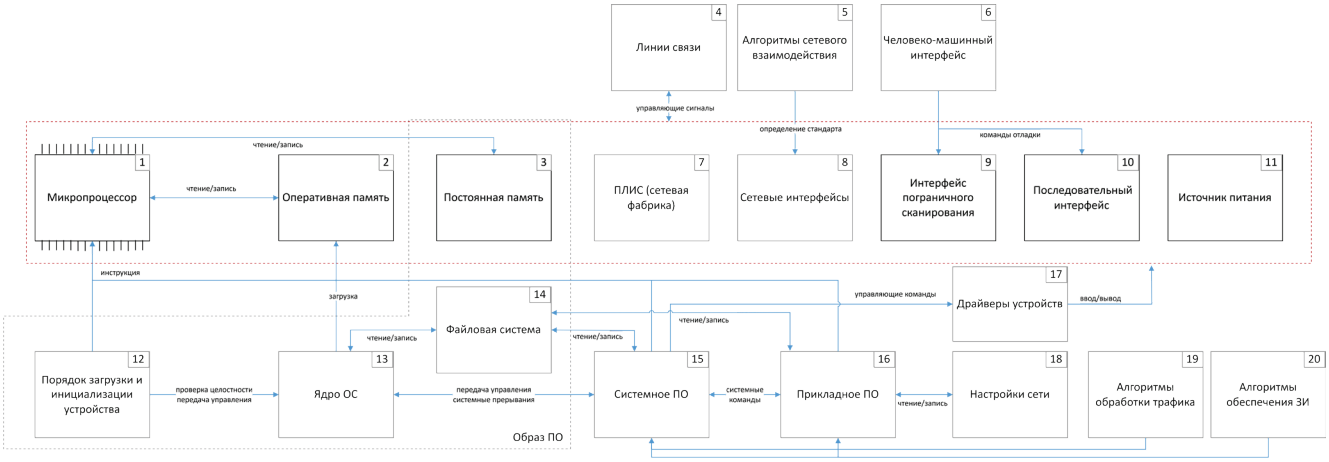


Рисунок 18 — Архитектурная модель КЭ СУ

Для моделирования АО на этом уровне абстракции используются графико-логические, принципиальные или другие блок-схемы. На Рисунке 19 представлена блок-схема 64-разрядного процессора MIPS, реализующего защищённую систему передачи данных на базе хеш-функций SHA-2 [89].

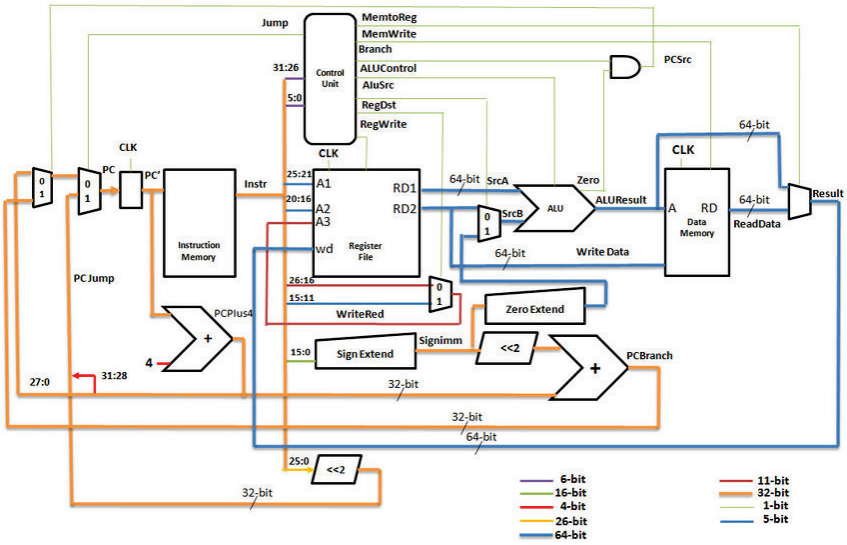


Рисунок 19 — Блок-схема процессора MIPS

Унифицированный язык моделирования (UML) является стандартом архитектурного моделирования для ПО. UML позволяет архитекторам, разработчикам и исследователям ПО использовать единые графические обозначения (Рисунок 20) не только для последовательного проектирования программных компонент и их связей, но и для проверки соответствия готовой реализации с проектом на основе обратного проектирования [90].

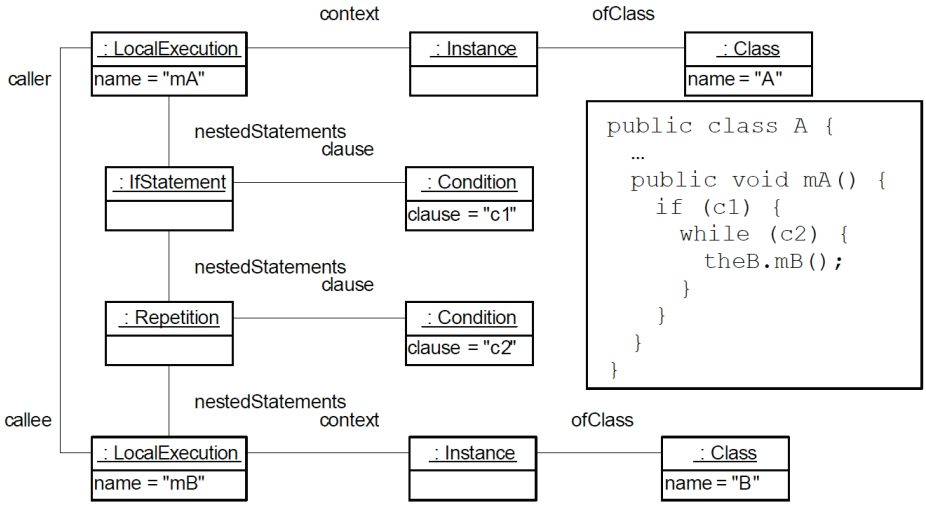


Рисунок 20 — Блок-схема трассы выполнения программного компонента

Модель оценки качества КЭ СУ зависит от требований, предъявляемых к системе. Процесс оценивания качества включает подпроцессы — сравнивания, верификации и валидации, которые подтверждают выполнение заданных требований. Отсюда следует, что необходимо создать иерархические метамодели требований. В соответствии с предлагаемой аналитической моделью оценки качества целесообразно разделить требования по характеристикам качества (Рисунок 21).

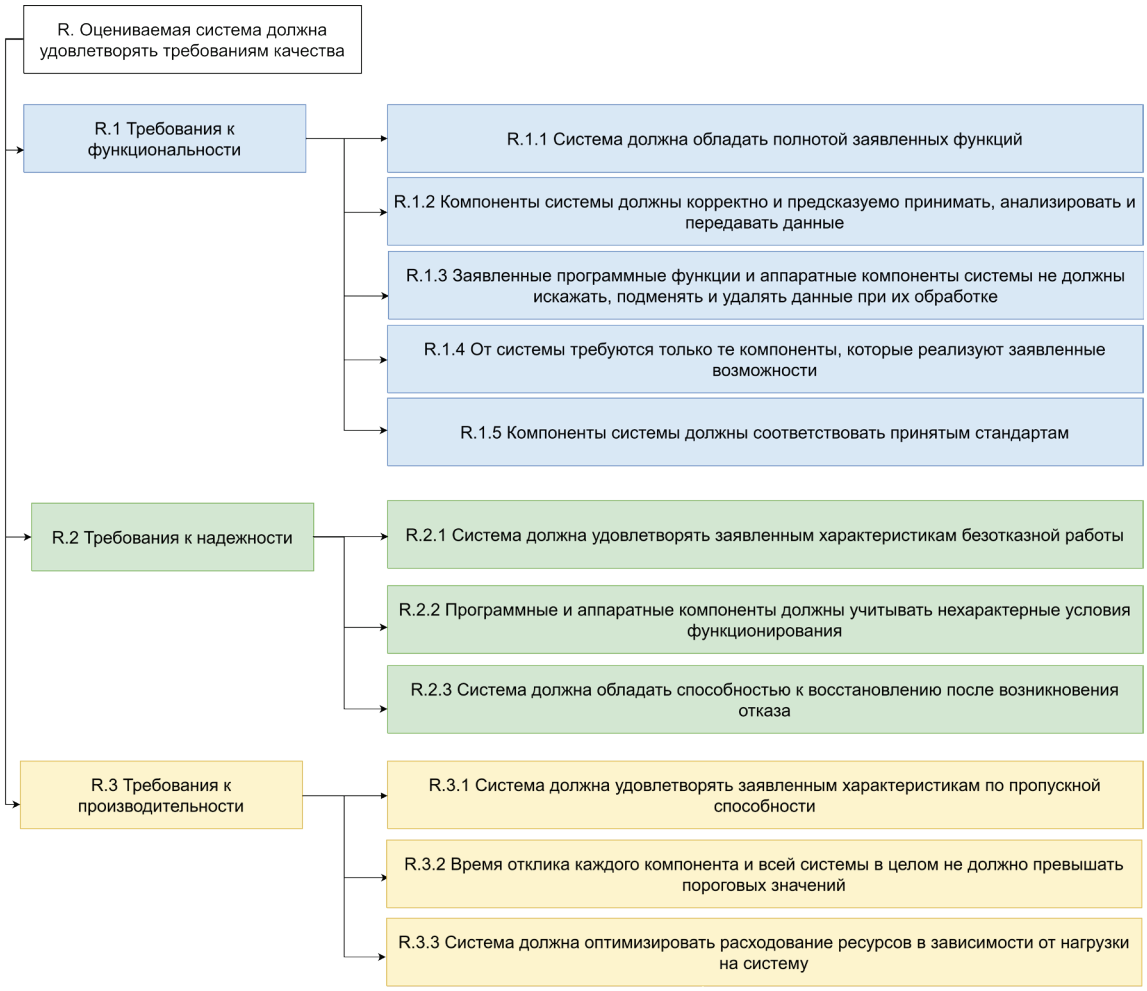


Рисунок 21 — Иерархическая метамодель требований к КЭ СУ

Согласно аналитической модели качества КЭ СУ и предлагаемой стратификации, исследуемая система разбивается на функции, которые она выполняет, и на компоненты, из которых она состоит. Формальное описание декомпозиций заключается в построении иерархических метамоделей с тем уровнем вложенности, который необходим для дальнейшей параметризации. Так как моделирование сопровождается отсутствием информации о системе (ограничения исследования), то полнота иерархических моделей функций

и компонент ограничена доступной информацией. Поэтому в процессе оценивания качества КЭ СУ при переносе ПО на ААП предлагается использовать методы обратного проектирования (метод анализа исполняемого кода и метод анализа аппаратных компонент), которые способны восстанавливать информацию о системе. Таким образом, применение методов обратного проектирования обеспечивает иерархические метамодели функций и компонент достаточной полнотой.

На Рисунке 22 представлена иерархическая модель функций КЭ СУ, главной функцией которого является обеспечение функционирования безопасного канала связи между элементами СУ.

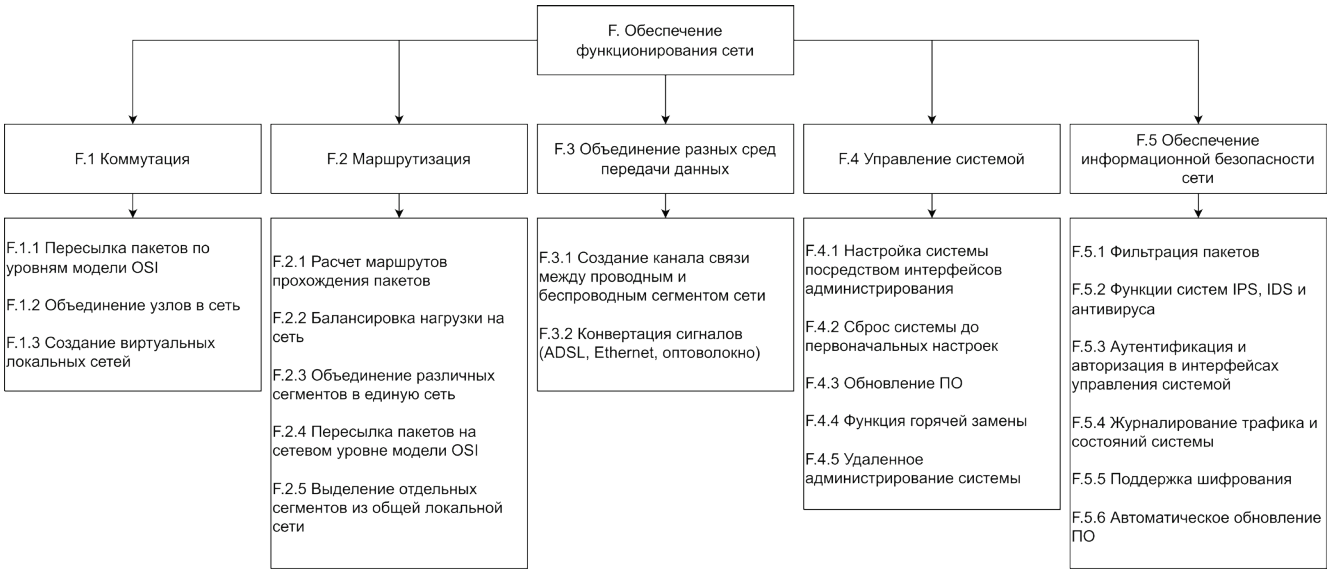


Рисунок 22 — Иерархическая метамодель функций КЭ СУ

Состав, уровень вложенности, детализацию иерархических метамоделей компонент рассматриваемой КЭ СУ необходимо с целью оценивания качества согласовывать с онтологическими моделями, требованиями и функциями. На Рисунке 23 представлен пример иерархической метамодели компонент КЭ СУ.

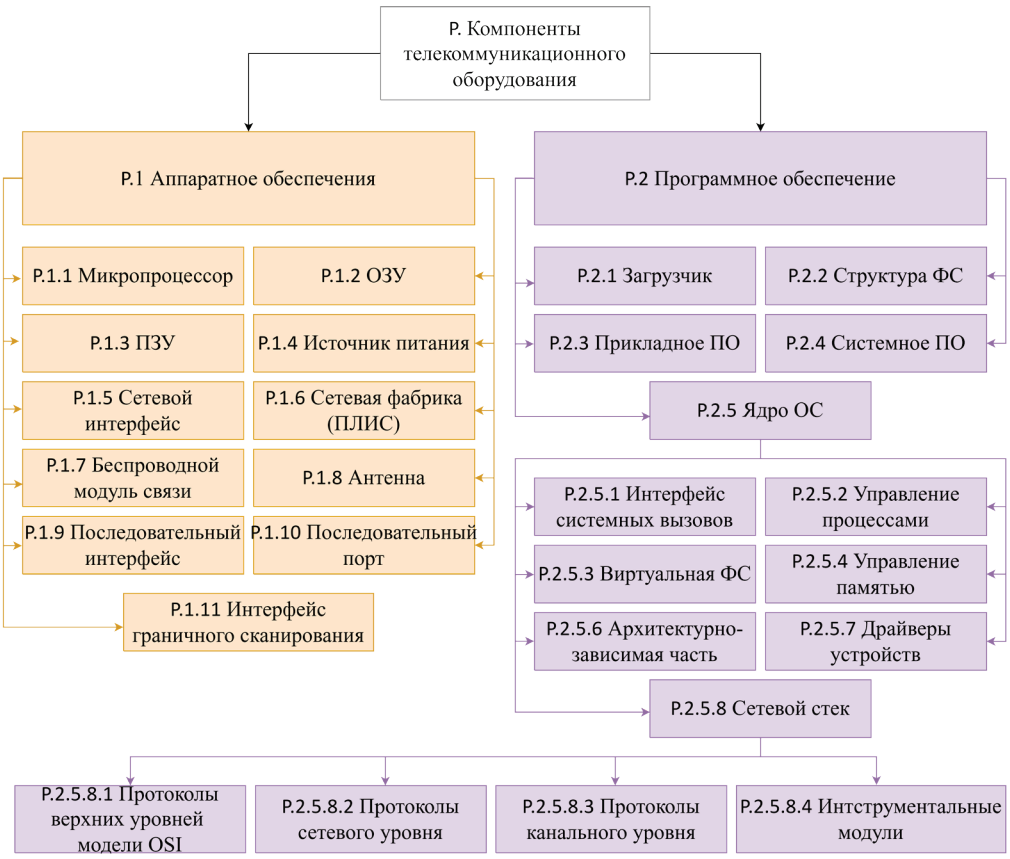


Рисунок 23 - Иерархическая метамодель компонент КЭ СУ

Метамодели требований, функций и компонент КЭ СУ позволяют составить параметризированные таблицы сущностей с количественными показателями (в аналитической модели являются ЭПК), а также определить типы связей между наиболее важными компонентами.

2.1.3 Параметризация требований, функций и компонент КЭ СУ

Оценка базовых характеристик качества КЭ СУ сопровождается измерением и расчётом значений ЭПК. Определение перечня ЭПК связано с параметризацией метамоделей требований, функций и компонент.

На основе требований к качеству КЭ СУ (Рисунок 21) составляется параметризированная таблица требований (Таблица 4), где ЭПК заданы качественными, временными и числовыми атрибутами.

Таблица 4 – Параметризация требований

Иерархически упорядоченные индексы компонент требований	Имена иерархически упорядоченных компонент требований	Метрики/единицы измерения и символьные обозначения атрибутов		
		a1	a2	a3
R.1.1	Функциональная полнота	Количество реализованных компонент (F^c)	Количество компонент, заданных в требованиях (F^m)	-
R.1.2	Корректность	Количество реализованных компонент (F^c)	Количество компонент, заданных в требованиях (F^m)	-
R.1.3	Точность	Множество всех входных данных (D)	Сумма результатов индикаторной функции $F_i^c(D_i)$	Сумма результатов индикаторной функции $F_i^m(D_i)$
R.1.4	Целесообразность	Количество реализованных компонент (F^c)	Количество компонент, обеспечивающих функционирование системы (F^a)	-
R.1.5	Согласованность	Количество реализованных компонент (F^c)	Количество компонент, реализованных по стандартам (F^s)	-
R.2.1	Безотказность	Интервал времени безотказной работы (t_i)	Количество отказов (m)	Индекс отказа (i)
R.2.2	Устойчивость к ошибкам	Тип отказа (N^*)	Общее количество отказов (N)	-
R.2.3	Восстанавливаемость	Время на восстановление системы (t_i^r)	Количество отказов (m)	Индекс отказа (i)
R.3.1	Пропускная способность	Вероятность отказа обработки данных ($p_n(t)$)	Момент времени (t)	-
R.3.2	Время отклика	Среднее время ожидания заявки ($T_{ож}$)	Среднее временя обслуживания ($T_{об}$)	-
R.3.3	Ресурсоёмкость	Загруженность микропроцессора ($Cl(t)$)	Количество используемой памяти ($Vm(t)$)	Время выполнения (t)

Чтобы соотнести КЭ СУ с требованиями, задаются качественные и числовые атрибуты для метамодели компонент (Рисунок 23). В Таблице 5 представлена часть параметризированных компонент КЭ СУ.

Таблица 5 — Параметризация компонент

Иерархически упорядоченные индексы компонент требований	Имена иерархически упорядоченных компонент требований	Метрики/единицы измерения и символьные обозначения атрибутов		
		a1	a2	a3
P.1.1	Микропроцессор	Тактовая частота	Разрядность	Архитектура
P.1.2	ОЗУ	Скорость чтения/записи (Мбит/с)	Объём памяти (МБ)	Тип памяти
P.1.3	ПЗУ	Скорость чтения/записи (Мбит/с)	Объём памяти (МБ)	Тип памяти
P.1.4	Источник питания	Выходное напряжение	Выходная мощность	Тип ИП
P.1.5	Сетевой интерфейс	Скорость приёма/передачи данных	Количество интерфейсов	Тип интерфейса
P.1.6	Сетевая фабрика (ПЛИС)	Скорость линии	Объём буфера	Тип микросхемы
P.1.7	Беспроводной модуль связи	Скорость приема/передачи данных	Тип модуля	Количество модулей
P.1.8	Антенна	Количество антенн	Тип антенн	Частота приема/передачи
P.1.9	Последовательный интерфейс	Скорость приёма/передачи данных	Количество интерфейсов	Тип интерфейса
P.1.10	Последовательный порт	Количество портов	Тип порта	-
P.1.11	Интерфейс граничного сканирования	Количество интерфейсов	Тип интерфейса	-
P.2.1	Загрузчик	Размер занимаемой памяти	Время работы	Тип загрузчика
P.2.2	Структура ФС	Размер занимаемой памяти	Тип ФС	-
P.2.3	Прикладное ПО	Размер занимаемой памяти (МБ)	Разрядность	Тип ПО
P.2.4	Системное ПО	Размер занимаемой памяти (МБ)	Разрядность	Скорость обработки данных
P.2.5	Ядро ОС	Размер занимаемой памяти (МБ)	Разрядность	Тип ядра ОС

Анализ системы включает определение связей между функциями и компонентами, которые эти функции реализуют. Связность представляется в виде таблиц отношений с указанием типов, свойств и других характеристик связей.

В Таблице 6 представлена часть связей между функциями и компонентами с двумя видами связей: программная (×) и аппаратная (∗).

Таблица 6 — Связность функций и компонент

Функции системы F	Компоненты системы P							
	P.1.1	P.1.2	P.1.3	P.1.5	P.1.6	P.2.3	P.2.4	P.2.5
F.1.1		∗		∗	∗		×	×
F.2.1	∗			∗				×
F.2.2	∗	∗		∗	∗		×	×
F.3.2	∗			∗	∗			×
F.4.1			∗			×	×	
F4.3			∗			×	×	
F.4.4		∗	∗				×	
F.5.2		∗	∗				×	×
F.5.4			∗			×	×	×

Математическое представление ПК, иерархические метамодели требований и функций позволяют составить таблицу отношений каждого требования ко всем интересующим функциям системы, в ячейках которой находятся ЭПК, а каждое требование соответствует ПК (Таблица 7).

Таблица 7 — Отношения требований к функциям КЭ СУ

Требования системы R	Функции системы F					
	F.1.1	F.2.1	F.2.2	…	F_i	Расчёт ПК
R.1.1	0…1	0…1	0…1	0…1	0…1	$\sum_{i=1}^{N} F^c / \sum_{j=1}^{K} F^m$
R.1.2	0…1	0…1	0…1	0…1	0…1	$1 - (\lvert F^m \backslash F^c \rvert) / \lvert F^m \rvert$
R.1.3	0…1	0…1	0…1	0…1	0…1	$\sum_{i=1}^{\lvert F_m \rvert}((F_i^c(D_i) - F_i^m(D_i))/F_i^m(D_i))/\lvert F^m \rvert$
R.1.4	0…1	0…1	0…1	0…1	0…1	$\sum_{i=1}^{N} F^a / \sum_{j=1}^{K} F^c$
R.1.5	0…1	0…1	0…1	0…1	0…1	$\sum_{i=1}^{N} F^s / \sum_{j=1}^{K} F^c$
R.2.1	$m_{1.1}$	$m_{2.1}$	$m_{2.2}$	m_n	m_i	$\sum_{i=1}^{m} t_i / m$
R.2.2	$m_{1.1}$	$m_{2.1}$	$m_{2.2}$	m_n	m_i	$\frac{m^*}{m}$
R.2.3	$m_{1.1}$	$m_{2.1}$	$m_{2.2}$	m_n	m_i	$\sum_{i=1}^{m} t_i^r / m$
R.3.1	$p_{1.1}(t)$	$p_{2.1}(t)$	$p_{2.2}(t)$	$p_n(t)$	$p_i(t)$	$1 - p(t)$
R.3.2	$T_{1.1}^{\text{ож}}, T_{1.1}^{\text{об}}$	$T_{2.1}^{\text{ож}}, T_{2.1}^{\text{об}}$	$T_{2.2}^{\text{ож}}, T_{2.2}^{\text{об}}$	$T_n^{\text{ож}}, T_n^{\text{об}}$	$T_i^{\text{ож}}, T_i^{\text{об}}$	$T_{\text{ож}} + T_{\text{об}}$
R.3.3	$Cl_{1.1}(t)$, $Vm_{1.1}(t)$	$Cl_{2.1}(t)$, $Vm_{2.1}(t)$	$Cl_{2.2}(t)$, $Vm_{2.2}(t)$	$Cl_n(t)$, $Vm_n(t)$	$Cl_i(t)$, $Vm_i(t)$	$\{Cl(t), Vm(t)\}$

Таблица 7 делится на три логические части (выделены цветом). Каждая часть объединяет показатели для отдельной характеристики качества, таких как: функциональная пригодность (выделено синим), надёжность (выделено зеленым), производительность (выделено жёлтым).

2.2 Уточнение аналитической модели. Связь показателей качества с ошибками в компонентах системы

В разделе 2.1 представлены формулы расчёта базовых характеристик качества (функциональной пригодности (q_1), надёжности (q_2), производительности

(q_3)) и относящиеся к ним ПК. Так как, помимо общей систематической погрешности при оценивании, присутствует погрешность из-за неравнозначного влияния ошибок на систему и ошибок, возникающих при переносе ПО на ААП (аппаратно-зависимых ошибок), то представленные формулы не позволяют рассчитать реальную оценку при модернизации КЭ СУ. Таким образом, возникает неопределённость при расчётах оценки качества, которую необходимо устранить за счёт применения новых и усовершенствования существующих ЭПК.

Функциональная полнота определяется не только количеством реализованных и требуемых функций, но и содержанием реализованных функций. Если функция КЭ СУ состоит из множества программных компонент (программы, набор программных функций, отдельные программные процедуры и т. д.), которые содержат преднамеренные или случайные ошибки, то нельзя сказать о том, что функция реализована. В то же время, если совокупность программных ошибок не является критичной (функция КЭ СУ работает без сбоев), то отнести функцию к нереализованной нецелесообразно. Поэтому введём параметр «$\widehat{f_i^c}$-степень реализованности программной функции»:

$$\widehat{f_i^c} = erfc\left(\sum_{k=1}^{n} w_k^f\right), \tag{43}$$

где w_k^f — оценка влияния программной ошибки на функциональную пригодность.

Использование обратной функции ошибок Гаусса [91] обусловлено тем, что с одной стороны небольшое количество незначительных ошибок не сильно влияет на реализацию программной функции, а с другой — при наличии единственной критичной ошибки функцию нельзя считать реализованной. Тогда функциональная полнота записывается следующим образом:

$$a_1^f = \frac{\sum_{i=1}^{|F^c|} \widehat{f_i^c}}{|F^m|}, \tag{44}$$

где F^c — это множество реализованных функций, каждым элементом которого являются реализованные программные функции $\{f_1^c, \dots, f_n^c\}$; F^m —

множество функций, которые могут быть реализованы без ошибок на множестве программных функций $\{f_1^m, \dots, f_n^m\}$.

Корректность определяется соответствием реализованных функций требуемым. Функция F_i^c соответствует функции F_i^m, если множество её параметров $\{p_1^c, p_2^c, \dots, p_n^c\}$ соответствует множеству параметров $\{p_1^m, p_2^m, \dots, p_n^m\}$, заданных в требованиях для функции F_i^m, и находится в границах допустимых значений.

$$a_2^f = 1 - \frac{|F^m \backslash F^c|}{|F^m|} \tag{45}$$

Так же, как и в исходной модели, выделяют полную и частичную корректность.

Степень детализации реализованных функций F^c зависит от внешнего параметра F^m. Если требования к функциям описываются на высоком уровне абстракции (например, в системе требуется авторизация по физическому ключу), то реализованной функцией является подсистема, состоящая из программных и аппаратных компонент, которые реализуют данное требование. Если требования к функциям описываются достаточно детально (например, в системе требуется авторизация по протоколу LDAP), то реализованной функцией является подсистема, состоящая из программных функций, которые реализуют протокол LDAP.

Точность оценивается как сумма разницы значений индикаторных функций, которые зависят от входных данных:

$$a_3^f = \frac{\sum_{i=1}^{|F^m|}(I_i^c(D) - I_i^m(D))}{|F^m|}, \tag{46}$$

где D — это множество $\{d_1, \dots, d_x\}$ входных значений в программную функцию.

Пусть $Dr \in D$, такое, что индикаторная функция $I_i^m(d_x) = 1$, тогда

$$I_i^c(d_x) = \begin{cases} 1, \text{ если } d_x \in Dr \\ 0, \text{если } d_x \notin Dr \end{cases} \tag{47}$$

Индикаторная функция $I_i^c(D)$ в области формальной верификации программ определяется как «*корректность функции*» в логике Хоара.

Пусть $F_i(d_x, d_y)$ — требуемая функция с входными параметрами d_x и выходными d_y. Предусловие функции обозначим $F_{pre}(d_x)$, а постусловие — $F_{post}(d_x, d_y)$.

Целесообразность

$$a_4^f = \frac{\sum_{i=1}^{|F^a|} f_i^a}{|F^m|}, \tag{48}$$

где F^a — это множество значений избыточных $\{f_1^a, \dots, f_n^a\}$ реализованных программных функций.

$$f_i^a = \begin{cases} \widehat{f_i^c}, f_i^c \in F^m \\ 0, f_i^c \notin F^m \end{cases} \tag{49}$$

Согласованность

$$a_5^f = \frac{\sum_{i=1}^{|F^c|} \widehat{f_i^s}}{|F^c|}, \tag{50}$$

где $\widehat{f_i^s}$ — оценка стандартности *i*-й программной функции.

$$\widehat{f_i^s} = \begin{cases} \widehat{f_i^c}, f_i^c \in F^s \\ 0, f_i^c \notin F^s \end{cases}, \tag{51}$$

где F^s — множество стандартных программных реализаций.

Надёжность КЭ СУ определяется безотказностью (53), устойчивостью к ошибкам (54), восстанавливаемостью (55). Пусть система обладает множеством отказов $M = \{m_1, \dots, m_i\}$ с m^* уникальных типов отказов, которые вызваны ошибками в ПО. Тогда для расчётов ПК надёжности имеет место такая характеристика отказа в системе, которая зависит от программных ошибок (52).

$$\widehat{m_i} = \begin{cases} \sum_{k=1}^{n} w_k^d, \ \sum_{k=1}^{n} w_k^d < 1, \\ 1, \sum_{k=1}^{n} w_k^d > 1 \end{cases} \tag{52}$$

где w_k^d — оценка влияния программной ошибки на надёжность системы.

Безотказность

$$a_1^d = (\sum_{i=1}^{n} \left(\frac{T_i - min(T)}{max(T) - min(T)} \right)) / n, \tag{53}$$

где $T = \{T_1, \dots, T_i\}$ — множество наработок до отказа.

$$T_i = \frac{t_i^M + t_i^I}{\widehat{m_i}} \text{ при } \widehat{m_i} \neq 0,$$

где t_i^M — это время до инициализации ошибочного программного кода, а t_i^I — время инициализации и выполнения ошибочного программного кода.

Устойчивость к ошибкам

$$a_2^d = \frac{m^*}{\sum_{i=1}^{n} \widehat{m_i}} \text{ при } \widehat{m_i} \neq 0, \quad (54)$$

где m^* — количество типов отказов.

Восстанавливаемость

$$a_3^d = (\sum_{i=1}^{n} \left(1 - \frac{TR_i - min(TR)}{max(TR) - min(TR)}\right))/n, \quad (55)$$

где $TR = \{TR_1, \dots, TR_i\}$ — значение восстанавливаемости после i-го отказа.

$$TR_i = \frac{t_i^R + t_i^M}{\widehat{m_i}} \text{ при } \widehat{m_i} \neq 0, \quad (56)$$

где t_i^R — это время, за которое выполняется программный код при восстановлении, а t_i^M — время, которое требуется для вмешательства извне.

Метрики системы массового обслуживания (СМО) как модели ВС являются ЭПК, что позволяет рассчитать ПК производительности, такие как: пропускная способность (59), время отклика (62), ресурсоёмкость по памяти (63), ресурсоёмкость по процессору (64).

Расчёт (59) и (62) основан на моделях СМО с отказами. СМО с ограниченной очередью моделируют компоненты КЭ СУ, которые обладают буфером данных. Буферы данных ограничены аппаратными возможностями системы, поэтому СМО с неограниченной очередью не подходят для моделирования. На Рисунке 24 представлена функция обработки сообщений (данных), которая обладает ограниченным буфером.

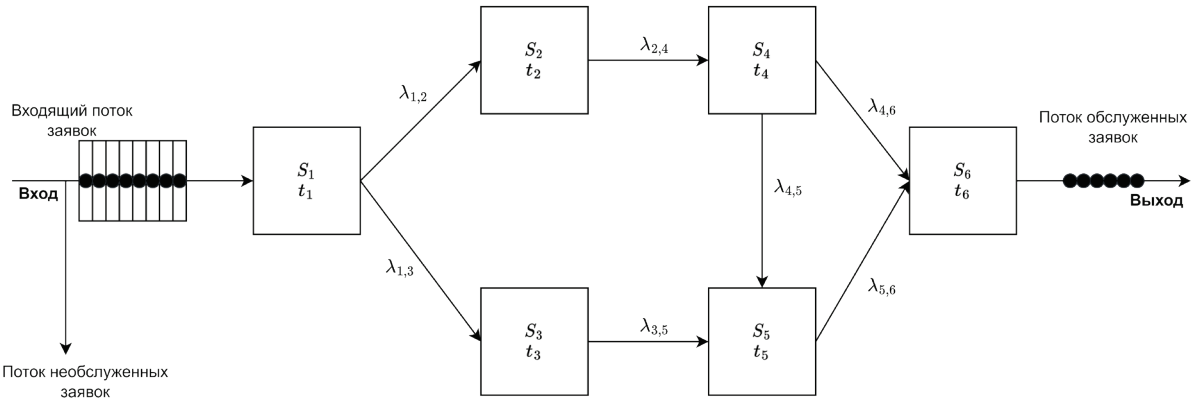


Рисунок 24 — Граф состояний функции обработки сообщений с ограниченным буфером

В результате исследований получены формулы, которые описывают расчёты пропускной способности и время отклика по отдельным программным компонентам системы.

Предельная вероятность при состоянии S_0:

$$p0_i = \frac{1-\rho_i}{1-\rho_i^{m+2}} \tag{57}$$

$$\rho_i = \frac{\lambda}{\hat{\mu}}, \tag{58}$$

где λ — интенсивность перехода из S_k в S_{k+1}, а $\hat{\mu} = f(w_k^p)$ — это функция интенсивности обработки сообщений, которая зависит от оценки влияния ошибки на производительность системы w_k^p.

Относительная пропускная способность

$$a_1^p = \sum_{i=1}^{n}\left(\frac{1-\rho_i^{m+1}}{1-\rho_i^{m+2}}\right) \tag{59}$$

Среднее число заявок, находящихся под обслуживанием:

$$Ls_{ij} = 1 - p0_i \tag{60}$$

Среднее число заявок, стоящих в очереди:

$$Lq_i = \rho_i^2 \frac{1-\rho_i^m(m+1-m\rho_i)}{(1-\rho_i)(1-\rho_i^{m+2})} \tag{61}$$

Время отклика

$$a_2^p = \sum_{i=1}^{n} \frac{Ls_i + Lq_i}{\lambda} \tag{62}$$

Модель программной функции — в виде одноканальной СМО без отказов — позволит рассчитать наиболее вероятные состояния функции, которые требуют анализа с точки зрения потребления ресурсов. После определения наиболее вероятных состояний выделяются те программные участки кода, которые отвечают за выделения памяти (на стеке и кучи), проводится подсчёт затрачиваемой памяти (V^m), а также определяется количество процессорного времени на обработку вероятного состояния (C^l).

Ресурсоёмкость по памяти

$$a_3^p = \frac{\sum_{i=1}^{n} V_i^m}{V}, \tag{63}$$

где V_i^m — объём затраченной памяти в i-ом программном компоненте, V — объём всей доступной памяти.

Ресурсоёмкость по процессору

$$a_4^p = \frac{\sum_{i=1}^{n} C_i^l}{C}, \tag{64}$$

где C_i^l — объём затраченного процессорного ресурса в i-ом программном компоненте, C — объём всего ресурса процессора.

Таким образом, введение оценок влияния программной ошибки на систему позволяет уточнить формулы расчёта в модели. Таблица 6 отношений требований к функциям преобразуется за счёт введения функциональных зависимостей (Таблица 8).

Таблица 8 — Математическое отношение требований к функциям системы

Требования к системе, *R*	**Функции системы, *F***			
	F_1	…	F_i	Расчёт ПК
$\boldsymbol{R_1}$	$\widehat{f_1^c} = erfc\left(\sum_{k=1}^{n} w_k^f\right)$	…	$\widehat{f_i^c} = erfc\left(\sum_{k=1}^{n} w_k^f\right)$	$a_1^f = \frac{\sum_{i=1}^{\lvert F^c \rvert} \widehat{f_i^c}}{\lvert F^m \rvert}$
$\boldsymbol{R_2}$	$\{p_1^c, p_2^c, \dots, p_n^c\}$	…	$\{p_1^c, p_2^c, \dots, p_n^c\}$	$a_2^f = 1 - \frac{\lvert F^m \backslash F^c \rvert}{\lvert F^m \rvert}$

Продолжение таблицы 8

Требования к системе, *R*	**Функции системы, *F***			
	F_1		F_1	Расчёт ПК
$\boldsymbol{R_3}$	$I_1^c(d_x) = \begin{cases} 1, \text{ если } d_x \in Dr \\ 0, \text{если } d_x \notin Dr \end{cases}$	...	$I_i^c(d_x) = \begin{cases} 1, \text{ если } d_x \in Dr \\ 0, \text{если } d_x \notin Dr \end{cases}$	$a_3^f = \frac{\sum_{i=1}^{\lvert F^m \rvert}(I_i^c(D) - I_i^m(D))}{\lvert F^m \rvert}$
$\boldsymbol{R_4}$	$f_1^a = \begin{cases} \widehat{f_i^c}, f_i^c \in F^m \\ 0, f_i^c \notin F^m \end{cases}$	...	$f_i^a = \begin{cases} \widehat{f_i^c}, f_i^c \in F^m \\ 0, f_i^c \notin F^m \end{cases}$	$a_4^f = \frac{\sum_{i=1}^{\lvert F^a \rvert} f_i^a}{\lvert F^m \rvert}$
$\boldsymbol{R_5}$	$\widehat{f_1^s} = \begin{cases} \widehat{f_i^c}, f_i^c \in F^s \\ 0, f_i^c \notin F^s \end{cases}$	...	$\widehat{f_i^s} = \begin{cases} \widehat{f_i^c}, f_i^c \in F^s \\ 0, f_i^c \notin F^s \end{cases}$	$a_5^f = \frac{\sum_{i=1}^{\lvert F^c \rvert} \widehat{f_i^s}}{\lvert F^c \rvert}$
$\boldsymbol{R_6}$	$\widehat{m_1} = \begin{cases} \sum_{k=1}^{n} w_k^d, \sum_{k=1}^{n} w_k^d < 1, \\ 1, \sum_{k=1}^{n} w_k^d > 1 \end{cases}$	...	$\widehat{m_i} = \begin{cases} \sum_{k=1}^{n} w_k^d, \sum_{k=1}^{n} w_k^d < 1, \\ 1, \sum_{k=1}^{n} w_k^d > 1 \end{cases}$	$a_1^d = (\sum_{i=1}^{n} \left(\frac{T_i - min(T)}{max(T) - min(T)} \right))/n$
$\boldsymbol{R_7}$				$a_2^d = \frac{m^*}{\sum_{i=1}^{n} \widehat{m_i}}$ при $\widehat{m_i} \neq 0$
$\boldsymbol{R_8}$				$a_3^d = = (\sum_{i=1}^{n} \left(1 - \frac{TR_i - min(TR)}{max(TR) - min(TR)} \right))/n$
$\boldsymbol{R_9}$	$\hat{\mu} = f(w_k^p)$	...	$\hat{\mu} = f(w_k^p)$	$a_1^p = \sum_{i=1}^{n} \left(\frac{1 - \rho_i^{m+1}}{1 - \rho_i^{m+2}} \right)$
$\boldsymbol{R_{10}}$				$a_2^p = \sum_{i=1}^{n} \frac{Ls_i + Lq_i}{\lambda}$
$\boldsymbol{R_{11}}$				$a_3^p = \frac{\sum_{i=1}^{n} V_i^m}{V}$
$\boldsymbol{R_{12}}$				$a_4^p = \frac{\sum_{i=1}^{n} C_i^l}{C}$

Приведение разнородных ПК к единой расчётной формуле для всех характеристик не представляется возможным. Значения разных ПК в рамках одной характеристики имеют различное влияние. Поэтому итоговая аналитическая модель оценки качества имеет многокритериальный характер (три критерия: функциональная пригодность, надёжность, производительность), а также ПК обладают весовыми коэффициентами.

Так как характеристики основаны на однородных метриках, то общая оценка является свёрткой всех показателей качества a_k. В силу того, что значимость показателей разная, то необходимо ввести весовые коэффициенты ω_i. Тогда расчёт характеристики q_1, q_2, q_3 записывается следующими формулами:

$$q_1 = \sum_{k=1}^{5} \omega_k^f * a_k^f =$$
$$= \omega_1^f * \frac{\sum_{i=1}^{|F^c|} \widehat{f_i^c}}{|F^m|} + \omega_2^f * \left(1 - \frac{|F^m \backslash F^c|}{|F^m|}\right) + \omega_3^f * \frac{\sum_{i=1}^{|F^m|} (I_i^c(D) - I_i^m(D))}{|F^m|} + \quad (44)$$
$$+ \omega_4^f * \frac{\sum_{i=1}^{|F^a|} f_i^a}{|F^m|} + \omega_5^f * \frac{\sum_{i=1}^{|F^c|} \widehat{f_i^s}}{|F^c|}$$

$$q_2 = \sum_{k=1}^{3} \omega_k^d * a_k^d =$$
$$= \omega_1^d * \left(\sum_{i=1}^{n} \left(\frac{T_i - \min(T)}{\max(T) - \min(T)}\right)\right)/n + \omega_2^d * \frac{m^*}{\sum_{i=1}^{n} m_i} + \quad (45)$$
$$+ \omega_3^d * (\sum_{i=1}^{n} \left(1 - \frac{TR_i - \min(TR)}{\max(TR) - \min(TR)}\right))/n$$

$$q_3 = \sum_{k=1}^{4} \omega_k^p * a_k^p =$$
$$= \omega_1^p * \sum_{i=1}^{n} \left(\frac{1 - \rho_i^{m+1}}{1 - \rho_i^{m+2}}\right) + \omega_2^p * \frac{\sum_{i=1}^{n} Ls_i + Lq_i}{\sum_{i=1}^{n} Ls_i' + Lq_i'} + \omega_3^p * \left(1 - \frac{\sum_{i=1}^{n} V_i^m}{V}\right) + \quad (46)$$
$$+ \omega_4^p * \left(1 - \frac{\sum_{i=1}^{n} C_i^l}{C}\right)$$

В итоге получена аналитическая модель качества ВС, которая позволяет рассчитать базовые характеристики (функциональная пригодность q_1, надёжность q_2, производительность q_3) КЭ СУ.

2.3 Анализ модели оценки качества КЭ СУ на адекватность

Прежде чем применять разработанную модель в процессе оценивания качества КЭ СУ, требуется установить правильность введённых параметров и адекватность получаемых оценок реальным свойствам объекта.

Так как за основу взята модель оценки сложной программной системы, то необходимо проверить возможность использования введённых параметров. Главный параметр, который вводится в модель, — это оценка влияния программной ошибки на систему. Отдельно от процесса оценивания качества такой параметр существует в разных предметных областях. При оценивании безопасности обрабатываемой информации в сложных программных системах используется показатель критической значимости ошибок. Для его расчёта применяются следующие методики: CVSS (от англ. Common Vulnerability Scoring System — «общая система оценки уязвимостей»), VPR (от англ. Vulnerability Priority Rating — «оценка приоритета уязвимости»), методика оценки уровня критической значимости уязвимостей ФСТЭК. Также существуют классификаторы ошибок в программном коде, где рассчитаны степени критичности для каждого типа ошибок (MITRE, CERT, PVS). Такие оценки являются параметрами в системах принятия решений и входят в модели рисков при создании и сопровождении КЭ СУ. В моделях оценки качества используются экспертные или регистрационные методы обнаружения ошибок, которые способны определить количество ошибок, но не их качество. Поэтому внедрение в модель сложного параметра влияния программной ошибки на систему представляется логически обоснованным развитием моделей оценки качества КЭ СУ.

Адекватность основывается на том, что модель обладает свойствами реальной оценки качества КЭ СУ, а именно:

1. Модель обладает необходимой полнотой рассчитываемых характеристик качества.

Модель включает те характеристики качества КЭ СУ, которые обеспечивают переходы в системе оценивания. Переход к оцениванию остальных характеристик не имеет смысл в случаях, когда система функционально не готова

(функциональная пригодность), работа системы сопровождается систематическими сбоями (надёжность); скорость приёма, обработки и передачи данных не позволяет функционировать системе управления (производительность). Расчёт базовых характеристик позволяет определить базовую оценку качества КЭ СУ, что является необходимой полнотой в системе оценивания.

2. Параметры модели измеряемы.

Наиболее адекватной представляется та модель, в которой существует возможность использовать значения из моделируемого объекта. Так как предметом исследования являются модели оценки качества в условиях неопределённого состояния при переносе ПО на ААП, то целесообразно рассматривать только возможность измерения параметров по готовому образцу ПАК.

Все параметры модели делятся на внешние и внутренние. К внешним относятся множество требуемых функций F^m, множество входных значений в программные компоненты D, множество стандартных программных реализаций F^s, интенсивность потока заявок. Эти параметры не зависят от получаемого образца системы, поэтому они задаются на предварительном этапе оценивания.

Внутренние параметры модели измеряются по всем компонентам системы. При описании уточнённой модели качества КЭ СУ представлены механизмы измерения с помощью анализа исполняемого кода.

3. Существует необходимый объём входных данных для расчётов.

Проблема анализа сложных программных систем заключается в обработке огромного количества состояний, в которых может находиться система. Переход от функциональных компонент к программным с дальнейшим анализом потока управления и данных приводит к комбинаторному взрыву. Такая ситуация создаёт условия для невозможности обрабатывать все данные. Поэтому модель включает иерархические параметризированные метамодели, которые уточняют места сбора данных из готового образца системы, тем самым формируется необходимый объём входных данных.

4. Измеряемые параметры входят в область допустимых значений модели.

Основное ограничение модели — это невозможность расчётов при отсутствии ошибок или отказов в системе. Состояние программной системы,

в которой нет ошибок, является вырожденным случаем. В то же время, если такие ситуации встречаются, то такой системе выставляется наивысшая оценка качества по базовым характеристикам и происходит переход на следующий этап оценивания.

Второе ограничение модели состоит в том, что расчёты не имеют смысла если у системы отсутствуют вычислительные ресурсы по процессору и памяти. Такое состояние КЭ СУ не представляется возможным, так как противоречит основным принципам работы существующих вычислительных систем.

5. Оценка зависит от внутреннего устройства системы.

Предлагаемая модель, в отличие от существующих, строится на основе информации, полученной из системы. Это достигается благодаря применению методов обратного проектирования и позволяет выявлять ошибки не за счёт анализа входных и выходных данных, а путём анализа потока данных на всех этапах.

Приведённые аргументы в пользу адекватности модели оценки качества подтверждают её состоятельность. Модель является полной, целесообразной, все параметры модели считаемы, расчётная оценка в модели отражает реальную оценку КЭ СУ.

2.4 Выводы

Проведены исследования и разработана аналитическая модель оценки качества КЭ СУ, которая позволяет рассчитать показатели функциональной пригодности, надёжности и производительности. Моделирование КЭ СУ с помощью метода модельно-ориентированного системного инжиниринга позволяет выделить сущности системы (требования, функции, компоненты) и обеспечить расчётную часть модели необходимым объёмом входных данных. Ограничения связаны с особенностями реализации КЭ СУ и политикой доступа к информации сторонних производителей отдельных компонент или подсистем,

которые входят в состав КЭ СУ. К таким ограничениям могут относиться системы защиты от исследования, закрытые протоколы обработки данных, криптографические методы защиты информации. Хотя при должном ресурсе эти ограничения преодолеваются, но модель становится не применима из-за нарушения баланса затрат к результату.

Совершенствование методов обратного проектирования, поиска и анализа ошибок в программном обеспечении за счёт внедрения разработанной аналитической модели способствует разработке метода оценки качества КЭ СУ системы управления в условиях переноса ПО на ААП.

ГЛАВА 3. РАЗРАБОТКА МЕТОДА МНОГОКРИТЕРИАЛЬНОЙ ОЦЕНКИ КАЧЕСТВА КОМПЬЮТЕРНЫХ ЭЛЕМЕНТОВ СИСТЕМЫ УПРАВЛЕНИЯ

Представленная в главе 2 аналитическая модель оценки качества включает три характеристики: функциональная пригодность, надёжность, производительность. Расчёт ПК согласно международным и национальным стандартам основан на измерении ЭПК (Рисунок 25).

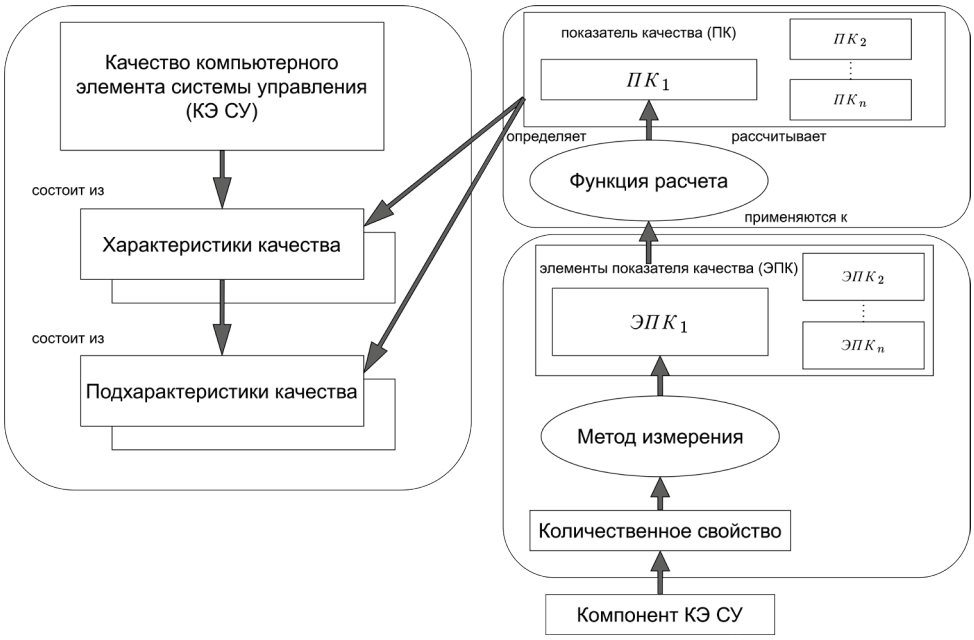


Рисунок 25 – Взаимосвязь свойств компонент КЭ СУ, ЭПК и ПК

Значения ЭПК определяются различными способами, наиболее популярные — это измерение метрик на готовом образце системы (метод «чёрного ящика»), исследование аналитических моделей, имитационное моделирование. В условиях переноса ПО на ААП подходящими методами считаются аналитические и имитационные. В то же время такие методы зависят от исходных предположений о состояниях системы в процессе её работы. Перенос ПО КЭ СУ на ААП является частью процесса модернизации, поэтому анализ получающегося образца исходит из того, что система не обладает новыми программными свойствами. Так как существует физическая разница между аппаратными платформами, то вновь созданная система обладает другими свойствами в программных компонентах, которые необходимо учитывать. Существующие методы позволяют рассчитать оценку качества путём измерения всех ЭПК

по готовому образцу, используя моделирование КЭ СУ и опорную стратификацию MBSE (Рисунок 26) [92].

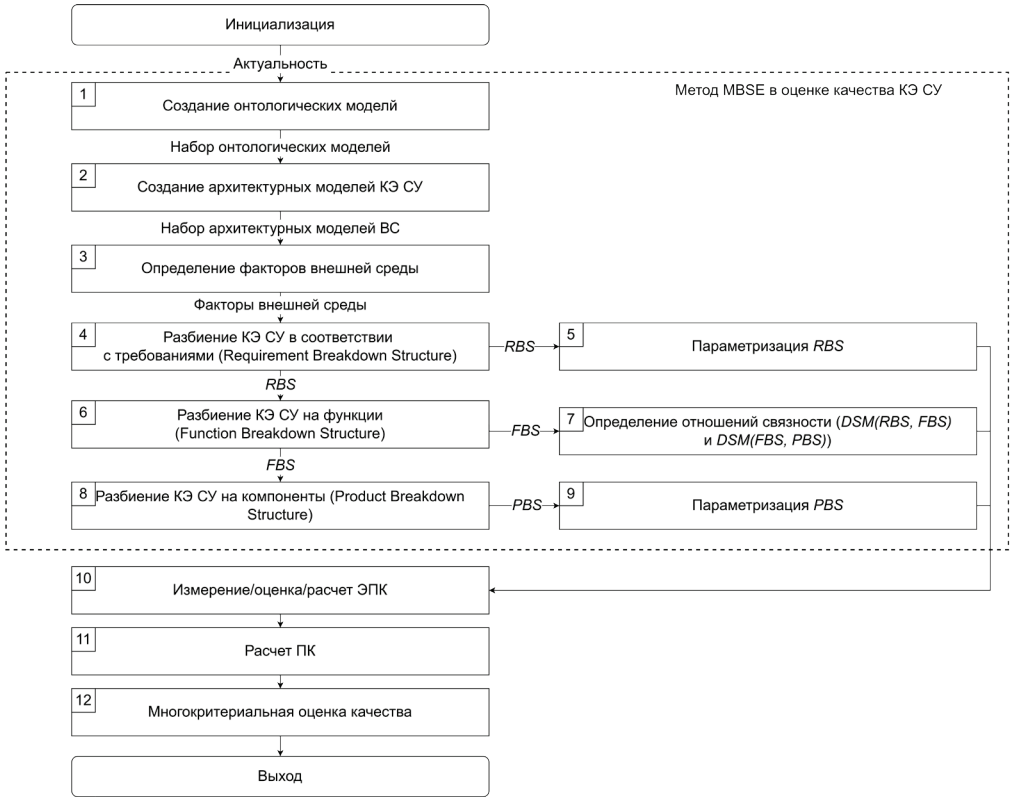


Рисунок 26 — Схема метода оценки качества КЭ СУ

Точность расчётов зависит от уровня детализации моделей отдельных компонент системы, который определяется целесообразностью. В свою очередь, применение методов обратного проектирования обеспечивает высокую точность при моделировании компонент системы. Так как в рамках исследования методы обратного проектирования применяются только к программным компонентам (ограничения исследований), то при построении метода используется только анализ исполняемого кода.

3.1 Алгоритм поиска ошибок в КЭ СУ за счёт анализа исполняемого кода программных компонент

Алгоритм поиска ошибок основан на априорных моделях программных компонент системы и их сравнении с моделями, построенными на основе анализа

исполняемого кода. В рамках данного исследования поиск ошибок разделён по характеристикам качества, поэтому используются соответствующие модели программных компонент. На Рисунке 27 представлен алгоритм поиска ошибок в программных компонентах готового образца системы.

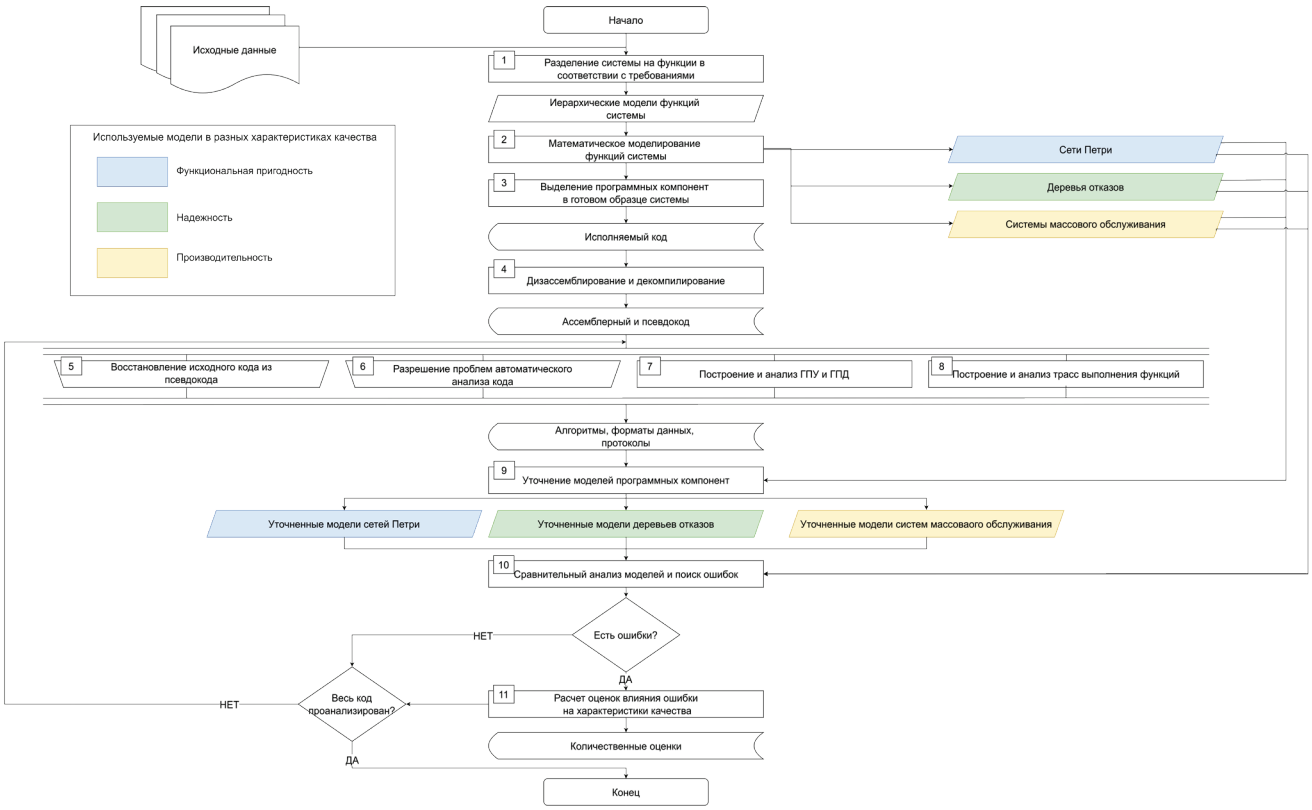


Рисунок 27 — Блок-схема алгоритма поиска ошибок

Алгоритм использует информацию о разделении системы в соответствии с функциями и компонентами (шаг 1). Так как моделирование функций (шаг 2) ограничено отсутствием информации о системе (факторы внешней среды), то полнота моделей FBS и PBS ограничена доступной информацией. Поэтому в процессе построения моделей FBS и PBS предлагается многократно использовать методы восстановления информации о системе (анализ исполняемого кода и восстановление информации об аппаратных компонентах) (шаги 3–9). Таким образом, применение методов обратного проектирования обеспечивает модели FBS и PBS достаточной полнотой для поиска ошибок путём сравнительного анализа априорных и уточнённых моделей (шаг 10). Сравнительный анализ позволяет выявить ошибку в коде, но предлагаемая модель оценки качества требует не количество ошибок, а степень их влияния на систему,

поэтому алгоритм включает расчёт оценки влияния каждой найденной ошибки (шаг 11).

Главное преимущество предлагаемого алгоритма состоит в построении и применении математических моделей, использующих информацию из исполняемого кода. В работе используется три класса математических моделей программных компонент: сети Петри, деревья отказов, системы массового обслуживания.

3.1.1 Моделирование программных функций по исполняемому коду сетями Петри

Моделирование программного кода сетями Петри — распространённая практика при разработке и верификации. Уровень моделирования выбирается в зависимости от требований к анализу получаемых моделей.

Программную часть ВС моделируют разными видами сетей Петри в зависимости от представления. К таким представлениям программного кода относят:

— граф потока управления (обыкновенные, ингибиторные, стохастические сети Петри);

— граф потока данных (цветные сети Петри);

— функции вычислений (алгебраические, ингибиторные сети Петри).

Так как для ингибиторных сетей Петри доказана полнота по Тьюрингу, то возможно точно моделировать программные компоненты КЭ СУ. В работе Зайцева Д. А. [93] рассматривается парадигма вычислений на сетях Петри. При использовании подобной парадигмы открывается возможность моделирования существующих систем посредством цепочки трансляций от исполняемого кода через язык ассемблера на язык сетей Петри. Это имеет большой смысл, когда нет доступа к исходных кодам и технической документации.

Сетью Петри называют двудольный ориентированный граф $\mathrm{N} = (P, T, F)$, где P — непустое множество позиций, T — непустое множество переходов, $F \subseteq (P \times T) \cup (T \times P)$ — непустое множество дуг, $\mathrm{P} \cap T = \emptyset$, каждая вершина сети $x, x \in P \cup T$ инцидентна хотя бы одной вершине другого типа. Ингибиторная сеть Петри дополняет обыкновенную сеть специальной функцией инцидентности $I_{in}: P \times T \to \{0,1\}$, которая вводит запрещающие (ингибиторные) дуги для тех пар (P, T), для которых $I_{in}(P, T) = 1$. Ингибиторные сети Петри позволяют строить модели разных уровней представления программного кода.

Пример сети Петри, которая описывает принцип работы программного компонента по графу потока управления, представлен на Рисунке 28.

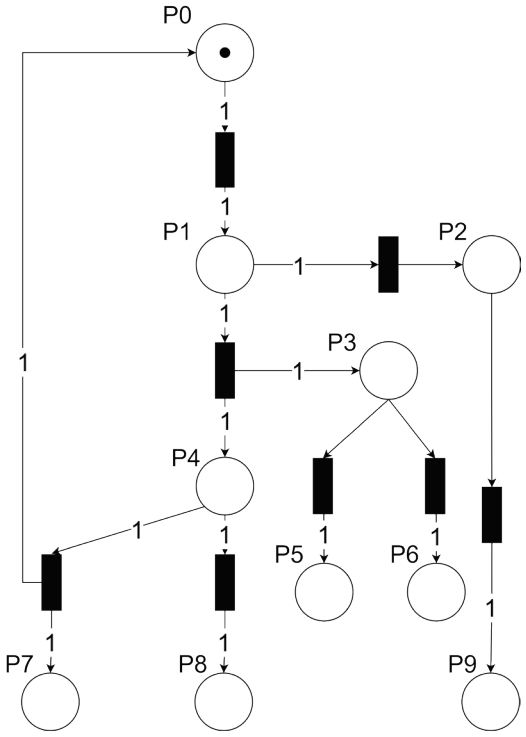
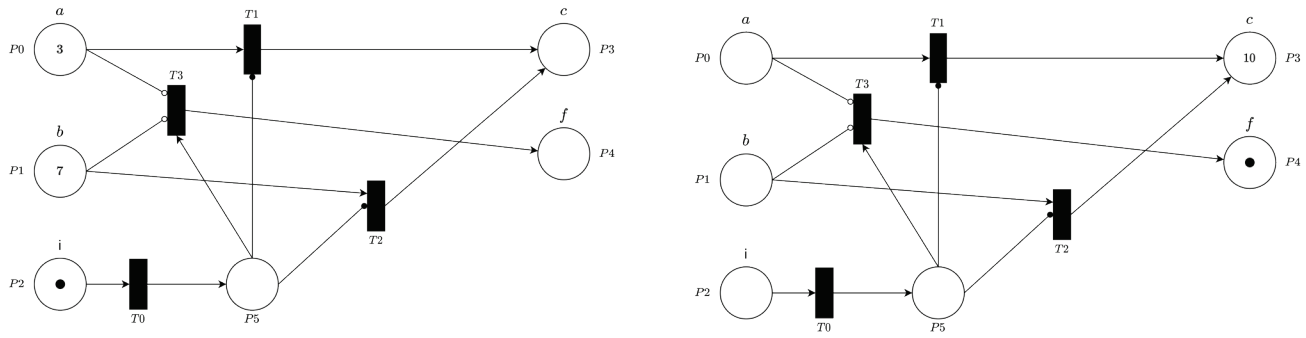


Рисунок 28 — Пример моделирования программной функции сетью Петри по графу потока управления

Построение детализированных моделей требует знаний о работе программных компонент или изучения исполняемого кода. Детализировать модель программных функций возможно до инструкций ассемблера. На Рисунке 29 представлена модель инструкции сложения целых чисел — *ADD*.



а б

Рисунок 29 — Модель сложения двух целых чисел

Такая модель описывает программный код на разных уровнях абстракций. В Таблице 9 представлены листинги сложения двух чисел на компилируемом языке высокого уровня, на интерпретируемом языке и на языке ассемблера.

Таблица 9 — Примеры программного кода

Язык программирования	Пример кода	Уровень абстракции

C/C++	`int a;`
	`int b;`
	`int c;`
	`a=3;`
	`b=7;`
	`c = a + b;`
		Средний

PHP	`<?php`
	`$A=3;`
	`$B=7;`
	`$C=$A+$B>`
		Высокий

Ассемблер x86	`MOV AX,3`
	`MOV BX,7`
	`ADD AX,BX`
		Низкий

Ассемблер MIPS	`li $t0,3`
	`li $t1,7`
	`add $t2,$t0,$t1`
		Низкий

Моделирование отдельных участков программного кода имеет смысл в тех случаях, когда необходимо провести сравнительный анализ ассемблерного и исходных кодов или локализовать ошибочный участок. В большей степени существует потребность в анализе базовых конструкций языка, которые характерны для всех программных компонент. К таким базовым конструкциям относятся программная процедура (Рисунок 30*а*), ветвление (Рисунок 30*б*), цикл (Рисунок 30*в*), распараллеливание (Рисунок 30*г*).

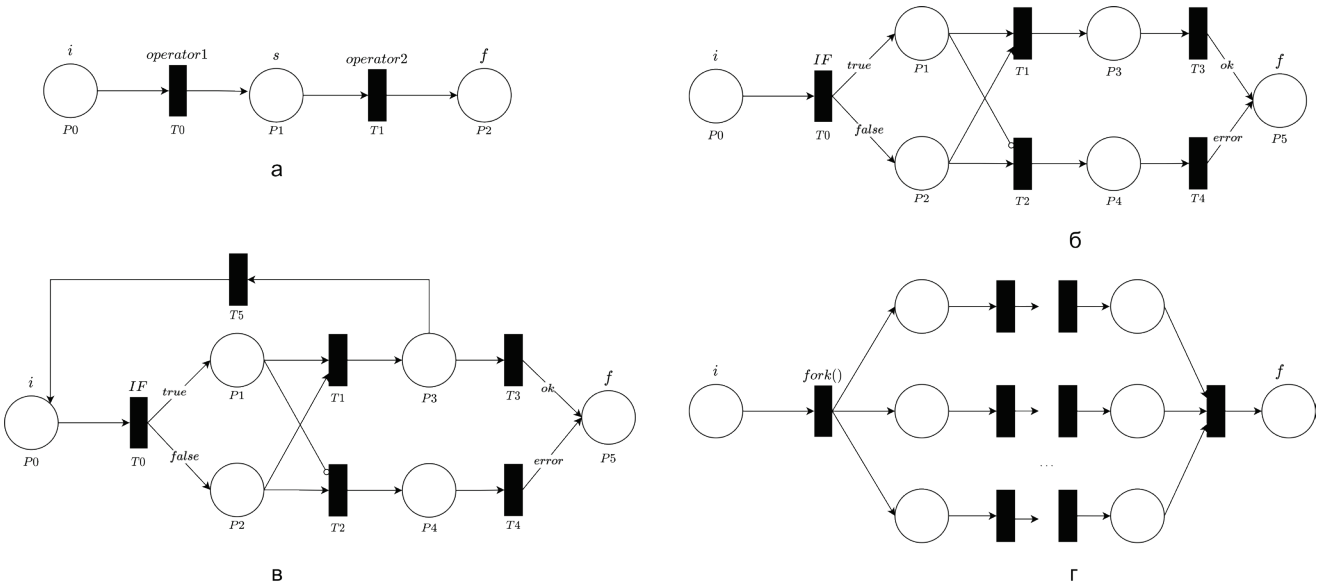


Рисунок 30 — Модели базовых конструкций программных компонент

Пусть КЭ СУ состоит из 21 функции (модель FBS представлена на Рисунке 22). Рассмотрим функцию «Аутентификация и авторизация» с точки зрения функциональной пригодности. Допустим, каждая функция КЭ СУ реализуется одной программной функцией. Листинг 1 описывает реализацию функции «Аутентификация и авторизация» на языке программирования С.

Листинг 1

```
int main (void)
{
 uint8_t len = 255;     // максимальное число пользователей
 for (uint8_t i = 0; i < len; i++) {     // цикл авторизации
  if (i == 0)     // условия того, что авторизация происходит впервые
  {
   initSettings ();         // функции первичной авторизации
  }
  else
  {
   ...
   if (strcmp(inuser, username) == 0)          // проверка имени пользователя
   {
    if (strcmp(inpass, password) == 0)          // проверка пароля
    {
     printf("User access\n");    // сообщение, что пользователь прошёл авторизацию
     continue;
    }
    else
    {
    printf("Wrong password\n");         // сообщение, что пользователь ввёл неверный пароль
    continue;
    }
  }
```

На основе имеющейся информации (исходный код, входные данные, выходные данные, поведение системы) строится априорная модель в виде сети Петри (Рисунок 31).

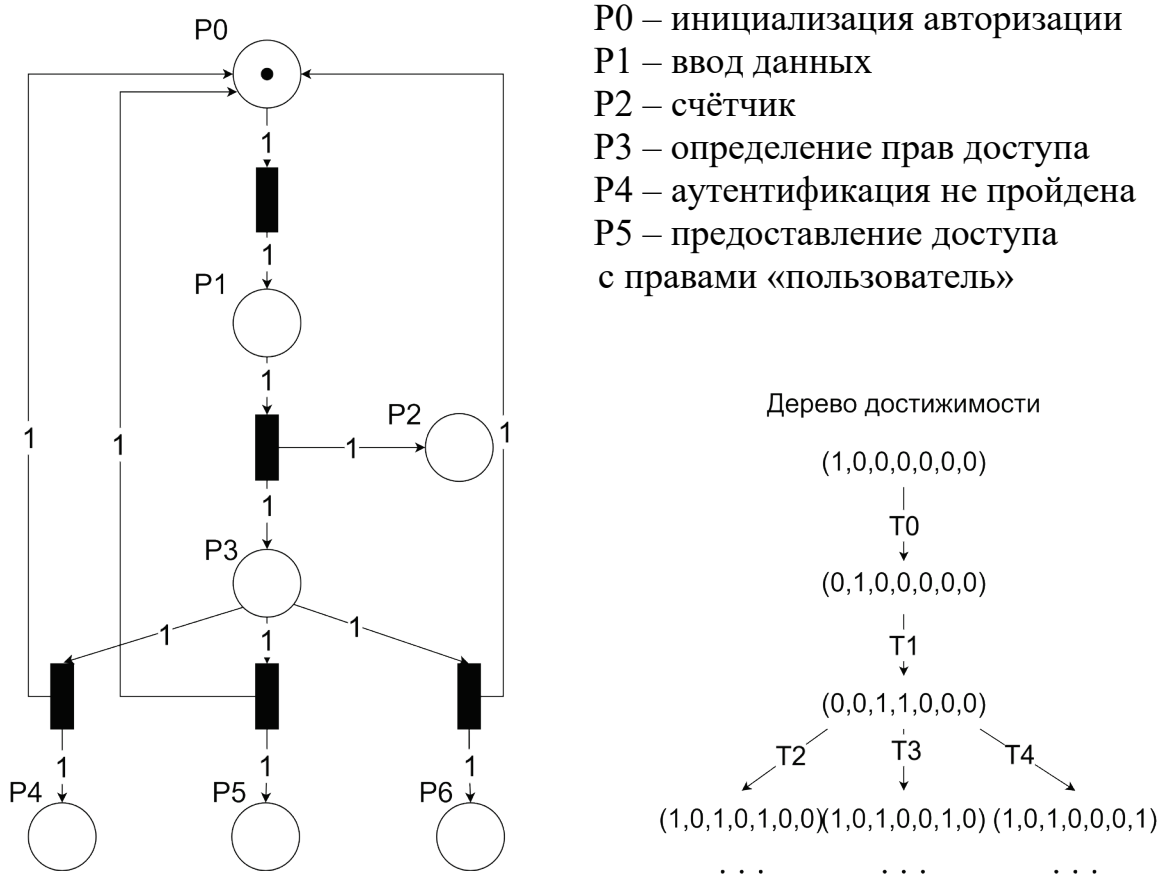


Рисунок 31 — Априорная модель сети Петри программной функции

Согласно предлагаемому алгоритму, требуется выделить функцию из исполняемого кода системы и уточнить априорную математическую модель. С помощью инструментов дизассемблирования (инструментов обратного проектирования) представляется возможным провести анализ скомпилированной программной функции. Граф потока управления дизассемблированной функции (аппаратная платформа x86_64) представлен на Рисунке 32.

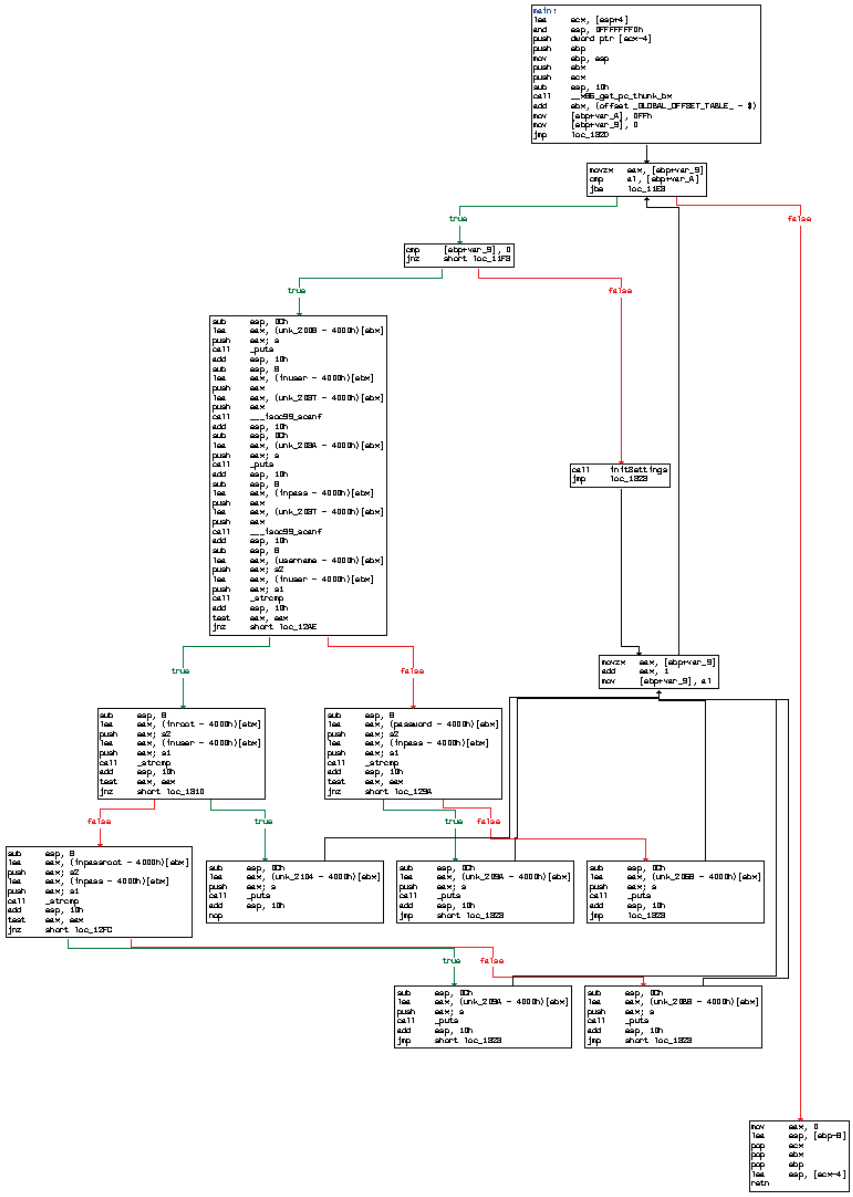


Рисунок 32 — Граф дизассемблированной программной функции

Анализ исполняемого кода позволяет выделить новые состояния и переходы, которые не учтены в априорной модели. В частности, реализация функции «Аутентификация и авторизация» содержит возможность авторизации без пароля в том случае, если пользователь заходит в систему впервые для настройки оборудования. Таким образом, априорная сеть Петри изменяется, у неё появляются новые свойства, а также анализ исполняемого кода может способствовать изменению в выборе класса сети, например, простая сеть Петри меняется на ингибиторную (Рисунок 33).

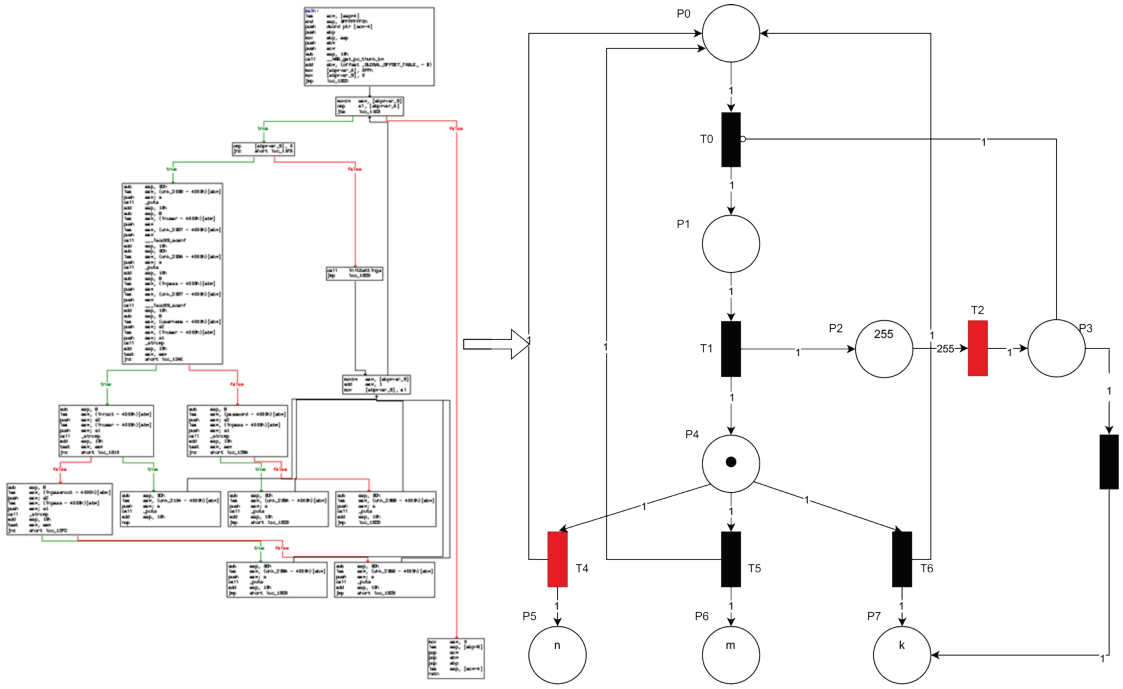


Рисунок 33 — Уточнённая сеть Петри

Сравнительный анализ априорной и полученной модели позволяет выявить ошибку целочисленного переполнения. Ошибка возникает, когда счётчик авторизаций уже равен 255. Подобные ошибки выявляются на этапах анализа исходного кода в том числе с помощью статических анализаторов кода. Предлагаемый метод уникален тем, что позволяет выявлять ошибки, которые возникают на одних аппаратных платформах и никак себя не проявляют на других (аппаратно-зависимые ошибки). Приведём пример исправления ошибки в Листинге 1 заменой условия и типа переменной (Листинг 2).

Листинг 2

```
int main (void)
{
 char i;// максимальное число пользователей
 while 1 {       // цикл авторизации
  if ( i < 1)      // условия того, что авторизация происходит впервые
  {
   initSettings ();        // функции первичной авторизации
  }
...
```

После компиляции исходного кода для платформ x86_64 и ARM функция «Аутентификация и авторизация» моделируется ингибиторными сетями Петри по исполняемому коду (Рисунок 34).

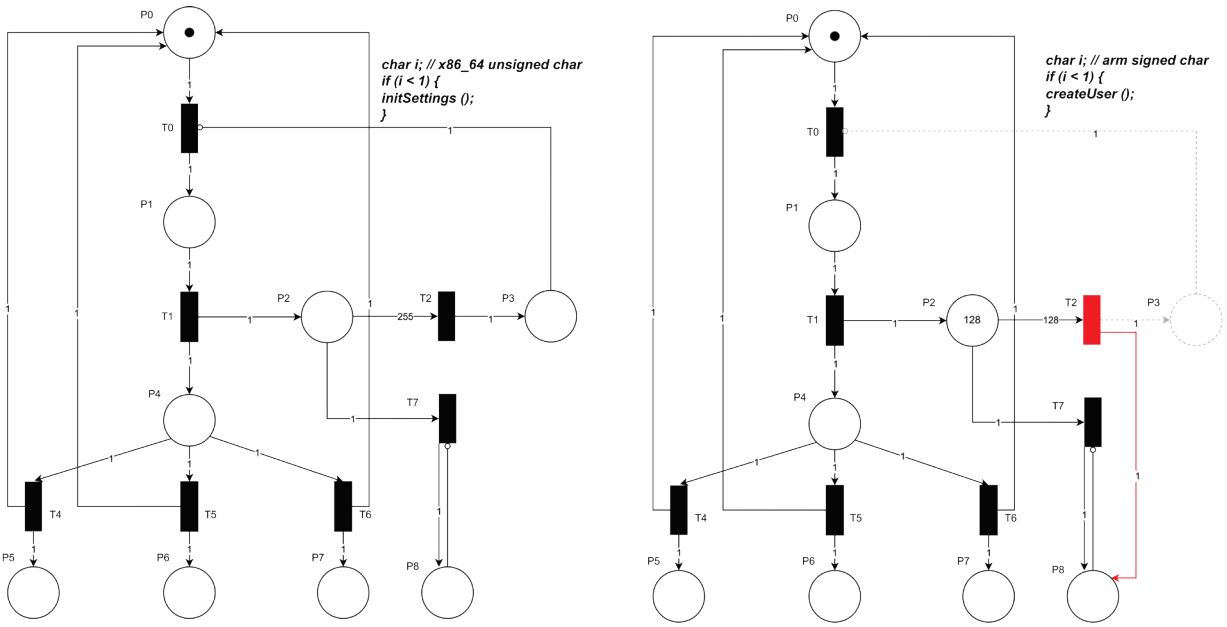


Рисунок 34 — Поиск аппаратно-зависимых ошибок в исполняемом коде

Сравнительный анализ моделей позволяет выявить ошибку целочисленного переполнения в исполняемом коде для архитектуры ARM. Ошибка возникает из-за разной трактовки одинаковых исходных текстов. Компилятор под аппаратную платформу ARM считает неопределённый тип *char* как знаковый, а компилятор по x86_64 — как беззнаковый. Статические анализаторы такую неопределённость разрешают за счёт выбора в пользу более распространённой платформы и поэтому пропускают аппаратно-зависимые ошибки.

Уровень функциональной пригодности КЭ СУ непосредственно зависит от наличия ошибок в реализованных функциях (F^c). Моделирование реализованных функций сетями Петри позволяет выявлять логические, функциональные, синтаксические, аппаратно-зависимые программные ошибки. Следовательно, становятся доступными измерения ЭПК, которые используются в аналитической модели оценки качества.

3.1.2 Модель надёжности с использованием деревьев отказов

Оценка надёжности КЭ СУ включает такие понятия, как «количество отказов N» и «тип отказа N_i^*». В процессе разработки системы отказы обнаруживаются

различными методами тестирования. Применение методов тестирования к готовому образцу осложняется отсутствием полноты знания о внутреннем устройстве системы, что ведёт к усугублению проблемы покрытия программного кода тестами. Применение методов тестирования к программному коду, который подготавливается к переносу на альтернативные аппаратные платформы, не имеет точных результатов, так как тестирование проводится на формальных моделях или тестовых образцах.

Предварительное разбиение системы на сущности (иерархические модели функций и компонент) позволяет формально описывать отказы в виде графико-логических схем. К таким схемам относятся деревья отказов, которые были рассмотрены в главе 1.

Определяющее значение в надёжности КЭ СУ имеет надёжность аппаратных H_i и программных P_i компонент. Более того, реализация программных компонент влияет на заявленные показатели надёжности аппаратных компонент. Отсюда следует, что построение деревьев отказа на основе программных компонент и с учётом аппаратной платформы позволяет выявлять вероятные цепочки событий, которые приводят к отказам.

При моделировании программно-аппаратных компонент выделяются:

A — базовые события;

B — промежуточные события;

C — отказ компонента системы;

SCE — отказ подсистемы;

SE — системный отказ.

На Рисунке 35 предлагается схема построения дерева отказов на основе программных функций ВС.

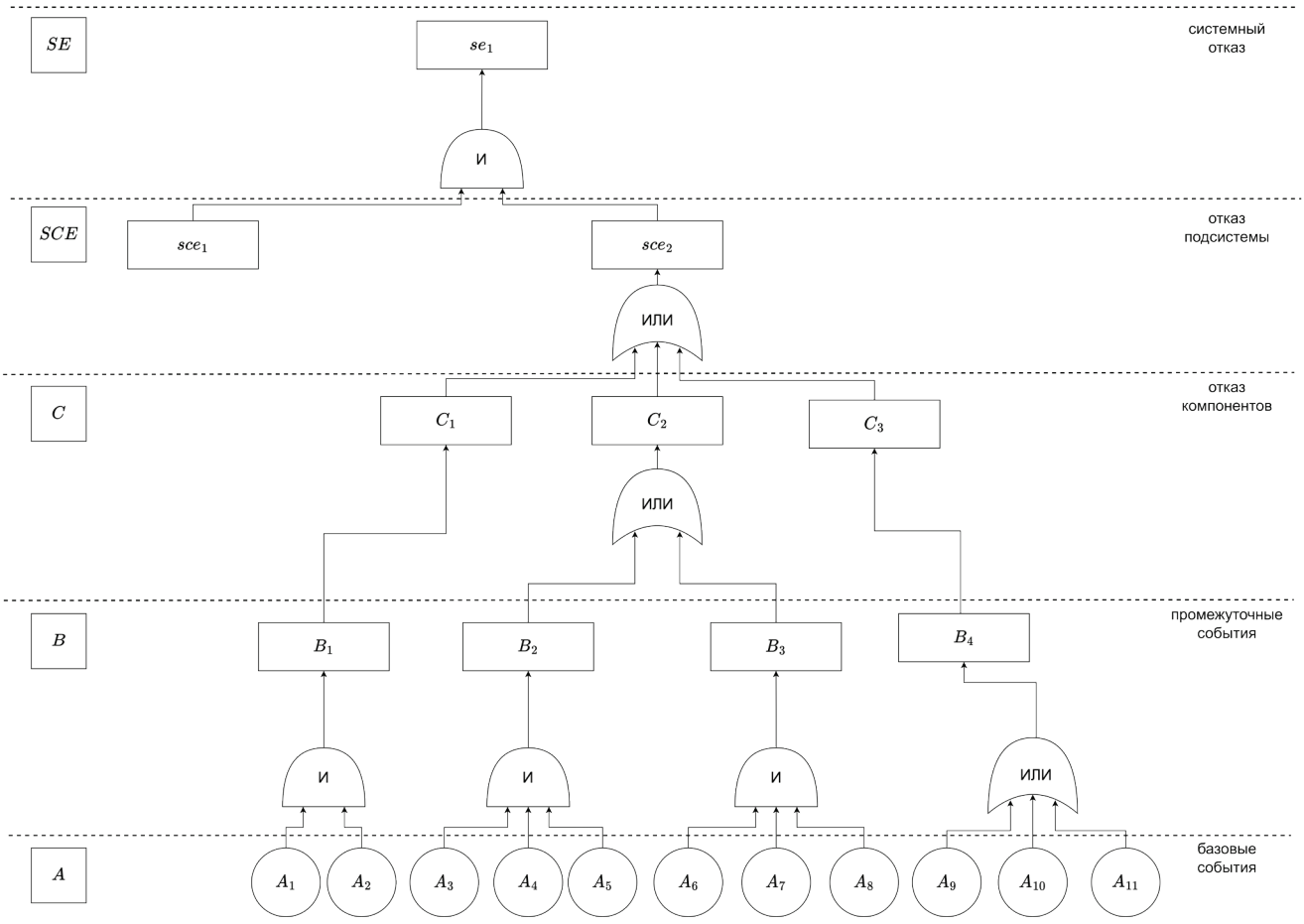


Рисунок 35 — Схема построения деревьев отказов

Пусть A — множество базовых событий A_i, вероятность наступления которых $P(A_i)$, тогда множество B промежуточных событий B_i рассчитывается как произведение вероятностей базовых событий, входящих в логический оператор «И».

$$P(A_1 A_2 \dots A_n) = \prod_{i=1}^{n} A_i \tag{65}$$

Вход базовых событий в логический оператор «ИЛИ» определяет вероятность промежуточного события по формуле суммы конечного числа совместных событий.

$$P\left(\sum_{i=1}^{n} A_i\right) = \sum_{i} P(A_i) - \sum_{i,j} P\left(A_i A_j\right) + \sum_{i,j,k} P\left(A_i A_j A_k\right) + \dots + (-1)^{n-1} P(A_1 A_2 \dots A_n), \tag{66}$$

где суммы распространяются на различные значения индексов i, j, k и их комбинаций.

В ходе действий по аналогии рассчитываются вероятности отказов компонент $P(C_i)$, отказов подсистем $P(SCE_i)$ и вероятность системного отказа $P(SE_i)$. Представим формальные расчёты вероятностей для построенного дерева отказа (Рисунок 14).

Дано:

Множество базовых событий $A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}\}$ с известными вероятностями срабатывания $P(A_i)$.

Для промежуточных событий определены логические правила:

ЕСЛИ A_1 И A_2 ТО B_1

ЕСЛИ A_3 И A_4 И A_5 ТО B_2

ЕСЛИ A_6 И A_7 И A_8 ТО B_3

ЕСЛИ A_9 ИЛИ A_{10} ИЛИ A_{11} ТО B_4

Правила для отказа компонент:

ЕСЛИ B_1 ТО C_1

ЕСЛИ B_2 ИЛИ B_3 ТО C_2

ЕСЛИ A_4 ТО C_3

Правило для отказа подсистем:

ЕСЛИ C_1 ИЛИ C_2 ИЛИ C_3 ТО SCE_2

Правило наступления системного отказа:

ЕСЛИ SCE_1 И SCE_2 ТО SE_1

Рассчитать: вероятности отказов $P(C_i), P(SCE_i), P(SE_i)$.

Используя формулы расчёта для совместимых и независимых событий, получаем следующие:

$$P(C_1) = P(A_1)P(A_2)$$

$$P(B_2) = P(A_3)P(A_4)P(A_5)$$

$$P(B_3) = P(A_6)P(A_7)P(A_8)$$

$$P(C_2) = P(B_2) + P(B_3) - P(B_2B_3))$$

$$P(C_3) = P(A_9) + P(A_{10}) + P(A_{11}) - P(A_9A_{10}) - P(A_{10}A_{11}) - P(A_9A_{11}) + P(A_9A_{10}A_{11})$$

$$P(SCE_2) = P(C_1) + P(C_2) + P(C_3) - P(C_1C_2) - P(C_2C_3) - P(C_1C_3) + P(C_1C_2C_3)$$

$$P(SE_1) = P(SCE_1)P(SCE_2)$$

С увеличением количества событий сложность схемы увеличивается, что усложняет процесс вычисления вероятностей наступления отказов. Целесообразно исключать из расчётов невероятные события.

Построим априорную модель функции «Обновление ПО» с помощью дерева отказов (Рисунок 36).

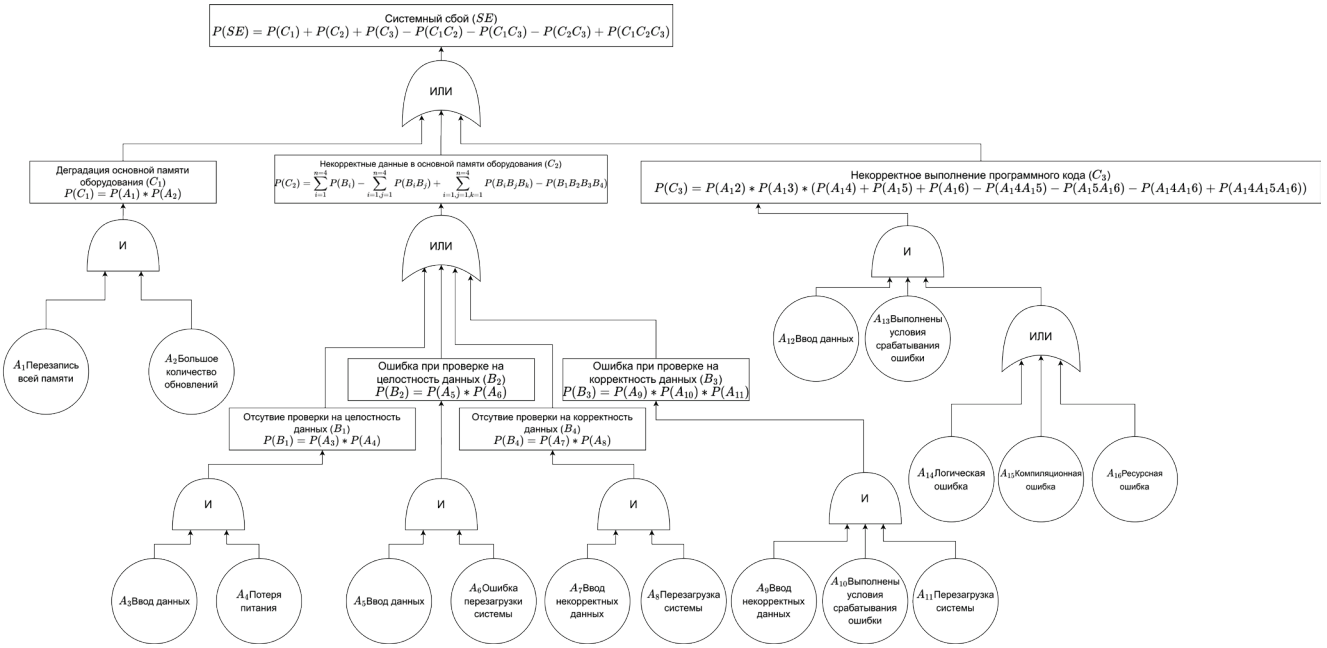


Рисунок 36 — Априорная модель дерева отказов для программной функции «Обновление ПО»

Создание априорной модели необходимо, так как это позволяет учитывать больше цепочек событий, которые гипотетически могут вызвать отказы различного уровня. Затем с помощью анализа исполняемого кода исключаются те события, которые не возникают в системе. На Рисунке 37 показаны дизассемблированный код и граф потока управления программных компонент, реализующих функцию системы «Обновление ПО», а также указаны базовые и промежуточные события, вероятность наступления которых равна нулю.

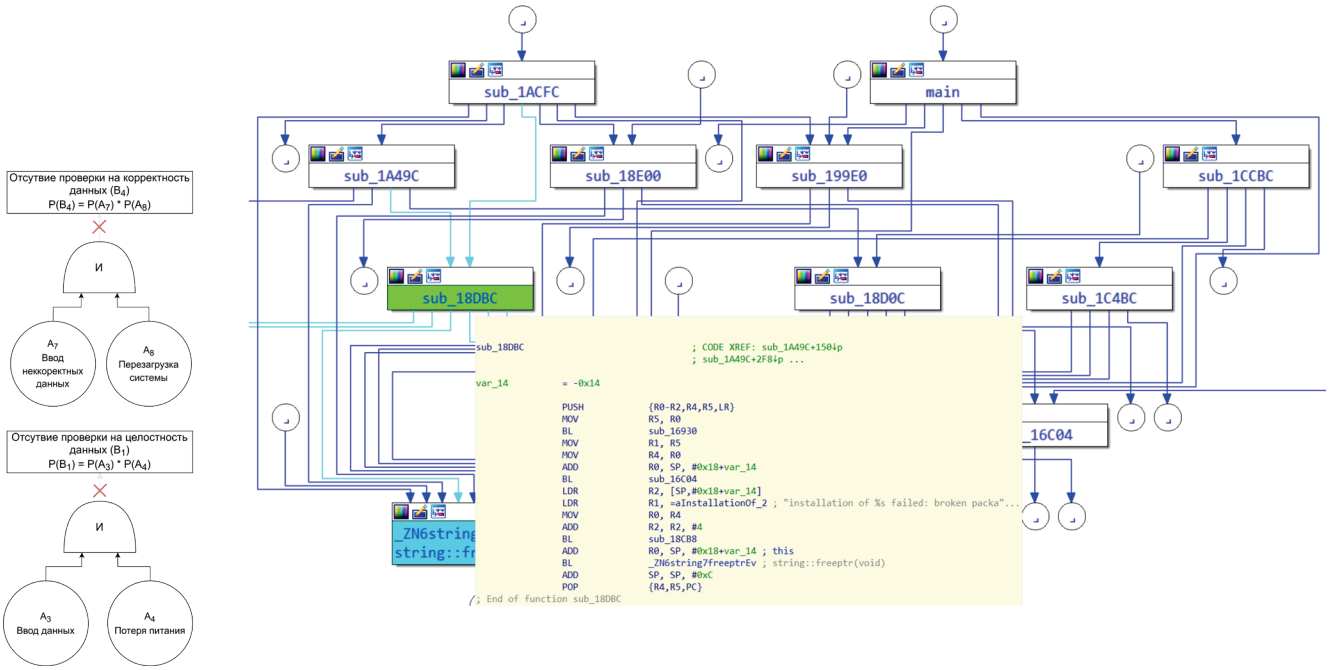


Рисунок 37 — Дизассемблированный код функции «Обновление ПО»

Выводы о том, что одновременное наступление событий «ввод некорректных данных» и «перезагрузка системы» или «ввод данных» и «потеря питания» вызовет системный сбой невероятно, так как это следует из анализа ассемблерного кода. Подтверждено наличие проверок корректности структуры файла с обновлениями ПО, а также установлено, что запись ПО во внутреннюю память происходит после загрузки всех данных в оперативную память. Следовательно, выявление невероятных событий в априорной модели значительно упрощает дерево отказов и сокращает расчёты (Рисунок 38).

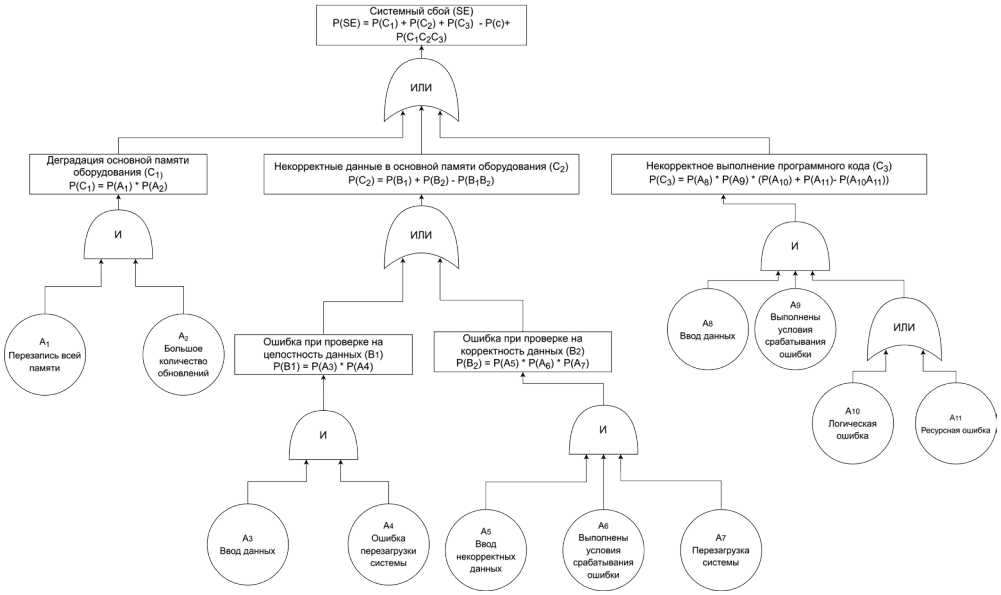


Рисунок 38 – Уточнённое дерево отказов функции «Обновление ПО»

Ключевым параметром при моделировании функций КЭ СУ с помощью деревьев отказа выступает вероятность отказа. На основе априорной информации о реализации и статических данных о программных ошибках строится гипотеза о таком состоянии функции, при котором существует вероятность наступления отказа. Точность расчёта этих вероятностей зависит от данных, которые соответствуют реальным состояниям системы. Применение методов анализа исполняемого кода обеспечивает расчёты данными, полученными непосредственно из системы.

3.1.3 Представление функций ВС в виде СМО

Оценка производительности КЭ СУ проводится измерительными методами в тех случаях, когда имеется образец системы. В случаях, когда система ещё не создана или ПО переносится на альтернативную аппаратную платформу, при оценке применяются математические модели. В области системного инжиниринга сложных программных систем на этапах разработки используется теория систем массового обслуживания.

Так как СМО позволяют моделировать потоки данных с различными свойствами, то применение моделей массового обслуживания для оценки производительности КЭ СУ сопровождается определением этих свойств в каждом компоненте. Теория СМО обеспечивает возможностью моделирования как всей системы, так и отдельных программных компонент (таких как ассемблерные инструкции, программные процедуры, программные функции, базовые конструкции программного кода).

Модель СМО позволяет оценить такие метрики, как:

— среднее число заявок, обслуживаемых в единицу времени;

— средняя доля заявок, обслуживаемых СМО в единицу времени, от среднего числа заявок, поступающих в СМО за то же время;

— средняя продолжительность периода занятости СМО;

— средняя доля времени, в течение которого СМО занята обслуживанием заявок;

— средняя доля времени, в течение которого СМО простаивает;

— среднее время ожидания обслуживания;

— среднее время пребывания заявки в СМО;

— вероятность отказа в обслуживании без ожидания;

— вероятность того, что прибывшая заявка немедленно будет принята к обслуживанию;

— среднее число заявок, находящихся в СМО;

— среднее число заявок в очереди;

— вероятность того, что число заявок в очереди превысит определённое значение.

Отсюда следует, что, используя метрики модели СМО в качестве ЭПК, рассчитываются ПК производительности (пропускная способность (a_{31}), время отклика (a_{32}), ресурсоёмкость памяти (a_{33}) и загрузка процессора (a_{34})). Ресурсоёмкость памяти и загрузка процессора оцениваются на основе времени обработки заявки на i-ом аппаратном компоненте (H_i).

На Рисунке 39 представлена модель программной функции, которая обрабатывает сообщения о рабочем состоянии системы. Сообщения информируют об ошибке или о предупреждении.

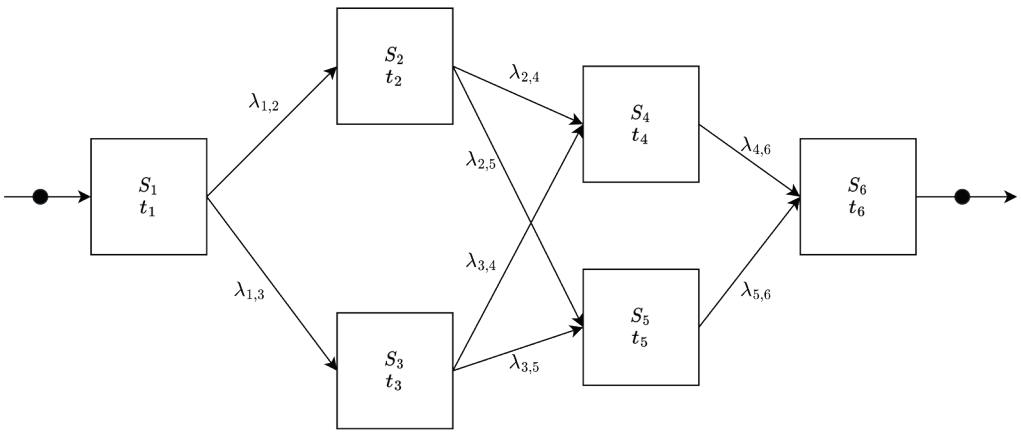


Рисунок 39 — Граф состояний функции обработки сообщений

Априорные вероятности переходов определяются с учётом того, что полная вероятность несовместных событий (заявка не может одновременно попасть в два перехода) равна 1. Отсюда следует, что вероятности переходов имеют следующие значения:

$$\mathrm{P}(S1 \to S2) = 1 - P(S1 \to S3)$$
$$\mathrm{P}(S2 \to S4) = 1 - P(S2 \to S5)$$
$$\mathrm{P}(S3 \to S4) = 1 - P(S3 \to S5)$$
$$\mathrm{P}(S4 \to S6) = 1$$
$$\mathrm{P}(S5 \to S6) = 1$$

Зная априорные вероятности переходов и время обработки заявки t_i, где i — это номер состояния, строится имитационная модель, на которой анализируется поведение функции в различных условиях [94].

Представленная программная функция имеет конечное число состояний S_i' с вероятностями $p_i(t)$. Представление приведённой программной функции как марковского процесса с дискретным состоянием и непрерывным временем показано на Рисунке 40 и описывается как система дифференциальных уравнений Колмогорова.

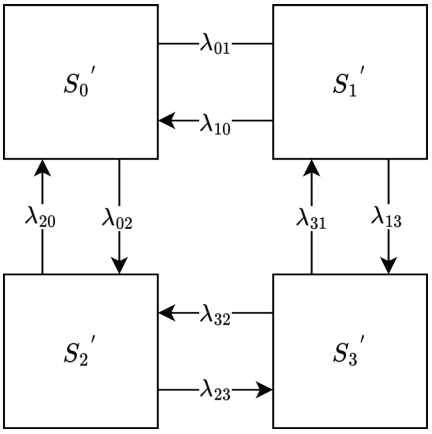


Рисунок 40 — Граф состояний программной функции в виде марковского процесса

Граф состояний в виде марковского процесса получен из графа состояния функции обработки сообщений (Рисунок 39), где:

$$S_0' : S_1 \to S_2 \to S_4 \to S_6$$
$$S_1' : S_1 \to S_2 \to S_5 \to S_6$$

$$S_2' : S_1 \to S_3 \to S_4 \to S_6$$

$$S_3' : S_1 \to S_3 \to S_5 \to S_6$$

Принимая время перехода системы из состояния S_i' в S_j' как Δt и в предельном случае $\Delta t \to 0$, получаем систему дифференциальных уравнений следующего вида:

$$\begin{cases} \dot{p}_0(t) = \lambda_{10} p_1(t) + \lambda_{20} p_2(t) - (\lambda_{01} + \lambda_{02}) p_0(t), \\ \dot{p}_1(t) = \lambda_{01} p_0(t) + \lambda_{31} p_3(t) - (\lambda_{10} + \lambda_{13}) p_1(t), \\ \dot{p}_2(t) = \lambda_{02} p_0(t) + \lambda_{32} p_3(t) - (\lambda_{20} + \lambda_{23}) p_2(t), \\ \dot{p}_3(t) = \lambda_{13} p_1(t) + \lambda_{23} p_2(t) - (\lambda_{31} + \lambda_{32}) p_3(t). \end{cases} \quad (67)$$

Расчёт пропускной способности и времени отклика основан на моделях СМО с отказами. В зависимости от реализации программной функции применяются:

для однопоточных систем:

— одноканальная СМО с ограниченной очередью;

— одноканальная СМО с ограничением на время ожидания в очереди;

для многопоточных систем:

— многоканальная СМО с ограниченной очередью;

— многоканальная СМО с ограничением на время ожидания в очереди.

СМО с ограниченной очередью моделируют программные компоненты КЭ СУ, которые обладают буфером данных. Буферы данных ограничены аппаратными возможностями системы, поэтому СМО с неограниченной очередью не подходят для моделирования. На Рисунке 41 представлена функция обработки служебных сообщений, которая обладает ограниченным буфером.

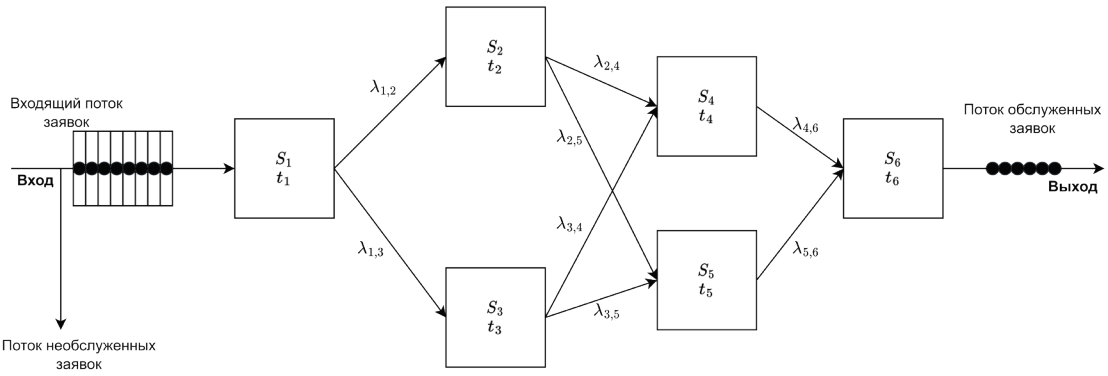


Рисунок 41 — Граф состояний функции обработки сообщений с ограниченным буфером

Одноканальная СМО с ограниченной очередью работает по схеме гибели и размножения с конечным числом состояний S_i'. На Рисунке 42 представлен размеченный граф системы.

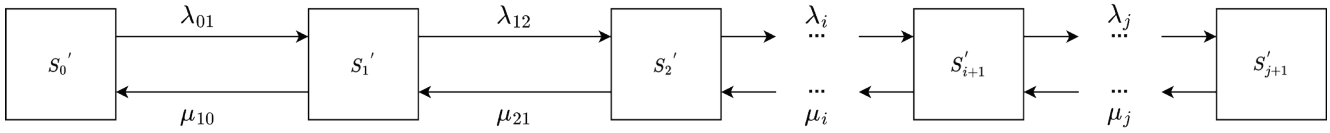


Рисунок 42 — Граф состояний программной функции, размеченный по схеме гибели и размножения

Определим следующие состояния системы:

S_0' — функция не задействована, сообщений на обработку нет;

S_1' — функция занята, сообщений в буфере нет;

S_2' — функция занята, одно сообщение в буфере;

S_{i+1}' — функция занята, i сообщений в буфере;

S_{j+1}' — функция занята, в буфере размером j находится j сообщений; таким образом, буфер заполнен и следующие сообщения будут отброшены, пока не освободится место в буфере.

Одноканальные СМО с ограниченной очередью имеют такие показатели эффективности, как вероятность отказа заявке в обслуживании $P_{\text{отк}}$ (68), относительная пропускная способность системы Q (69), абсолютная пропускная способность системы A (70); среднее число заявок, находящихся под обслуживанием $L_{\text{об}}$ (71); среднее время пребывания заявки под обслуживанием $T_{\text{об}}$ (72).

$$P_{\text{отк}} = \rho^{j+1}\frac{1-\rho}{1-\rho^{j+2}}, \tag{68}$$

где $\rho = \frac{\lambda}{\mu}$, j — состояние системы.

$$Q = \frac{1-\rho^{j+1}}{1-\rho^{j+2}} \tag{69}$$

$$A = \lambda\frac{1-\rho^{j+1}}{1-\rho^{j+2}} \tag{70}$$

$$L_{\text{об}} = 1 - p_0, \tag{71}$$

где p_0 — предельная вероятность состояния S_0.

$$T_{об} = \frac{L_{об}}{\lambda} \quad (72)$$

Построение точных моделей СМО для оценки производительности требует информации о внутреннем устройстве КЭ СУ. Поэтому априорная модель создаётся на основе анализа исполняемого кода и требует постоянного уточнения за счёт проведения статического и динамического анализа. Также для построения модели требуются технические характеристики аппаратных компонент.

Пусть функция КЭ СУ «Пересылка сетевых пакетов» реализована в отдельной программе, которая входит в состав встроенного программного обеспечения. Рассмотрим программную функцию проверки структуры сетевого пакета. Согласно алгоритму поиска ошибок, программную функцию необходимо дизассемблировать (Рисунок 43).

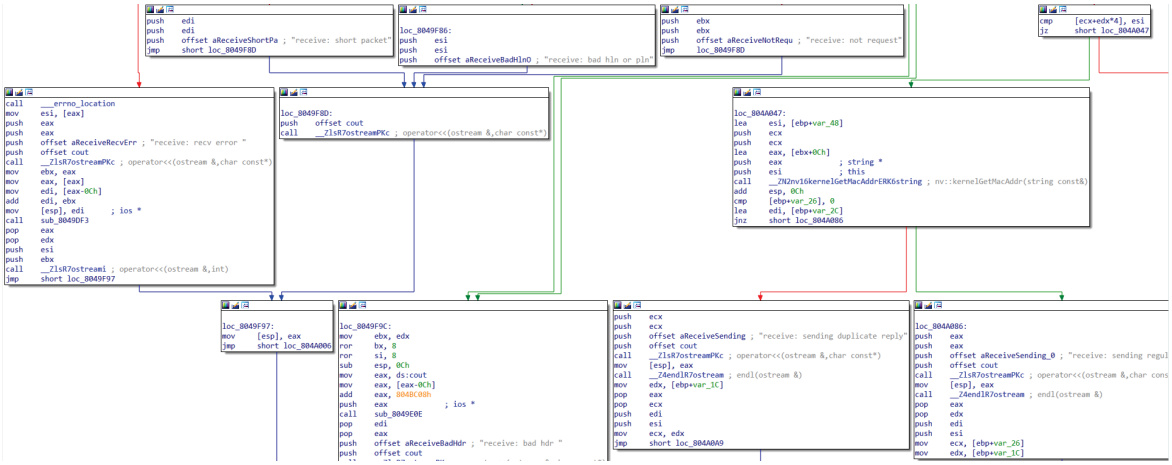


Рисунок 43 — Дизассемблированная программная функция проверки структуры сетевого пакета

На основе ассемблерного кода строится модель СМО с ограниченной очередью (Рисунок 44).

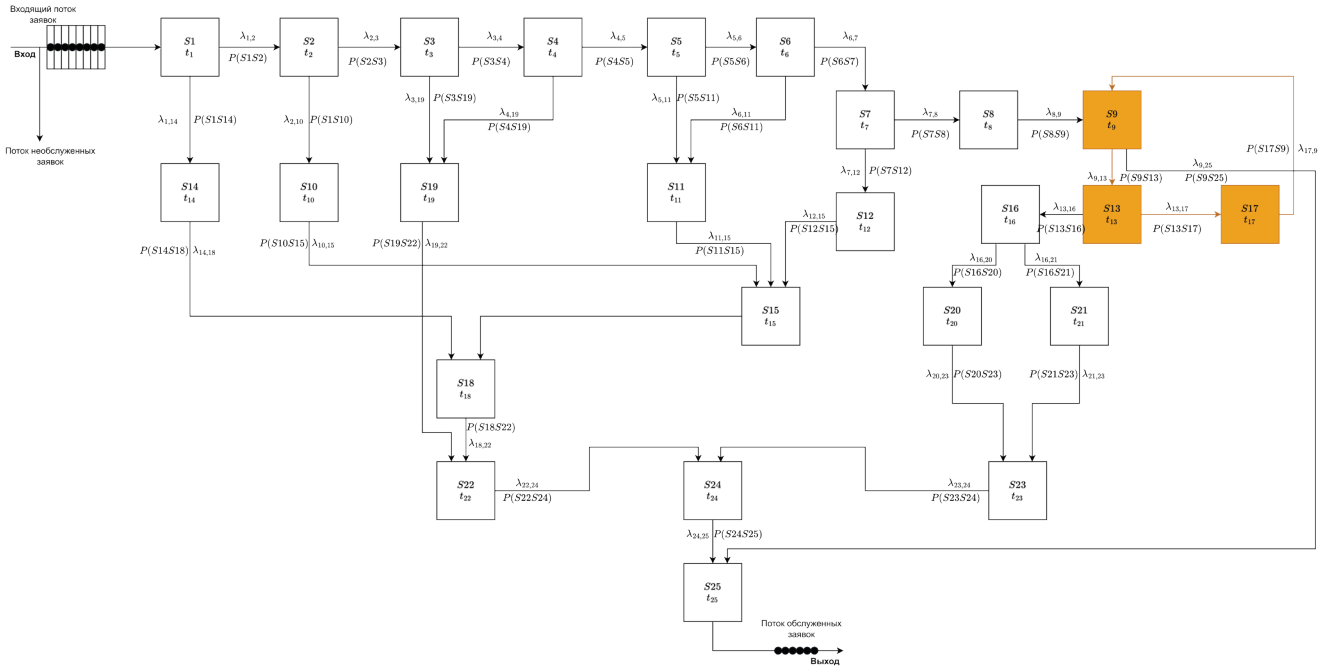


Рисунок 44 — Модель СМО программной функции проверки структуры сетевого пакета

Анализ ассемблерного кода обеспечивает модель СМО такими значениями, как:

— размер буфера;

— состояния программной функции;

— количество переходов из состояния;

— вероятность перехода из состояния;

— время обработки сообщений в каждом состоянии;

— интенсивность обработки сообщений;

— объём используемой основной памяти;

— объём используемой оперативной памяти;

— средняя загрузка процессора.

Размер буфера сообщений определяется количеством выделенной памяти в байтах. В моделируемой функции размер сообщения равен 42 байтам (Листинг 3), а размер буфера сообщений равен 64 Кб.

Листинг 3

```
push    ebp
mov     ebp, esp
push    edi
```

Продолжение листинга 3

```
push    esi
push    ebx
sub     esp, 40h
mov     ebx, [ebp+arg_0]
push    2Ah ; '*'       ; количество байт 0x2a = 42
lea     eax, [ebp+buf]
push    eax             ; buf
push    dword ptr [ebx+10h] ; fd
call    _read ; запись в буфер сообщения
add     esp, 10h
test    eax, eax
jns     short loc_8049F4B
```

Исходя из полученных данных, размер входящего буфера равен 1 560 сообщениям.

Состояния программной функции определяются программными процедурами (на Рисунке 39 процедуры обозначены как S_i). Каждая процедура имеет как минимум один переход в следующую, количество переходов определяется ассемблерной инструкцией (например, условный *(jz short loc_804A0B5*) или безусловный переход (*jmp loc_8049F8D*)). Вероятность переходов зависит от того, по какой ветви идет поток управления. Например, вероятность перехода из состояния $P(S_1S_2) = 0{,}2$, так как переход из процедуры S_1 в S_2 дополнительный и срабатывает только в случае обработки исключительной ситуации (*jns short loc_8049F48*). Компилятор специальным образом оптимизирует код так, чтобы было наименьшее количество «прыжков» по адресам (Рисунок 45).

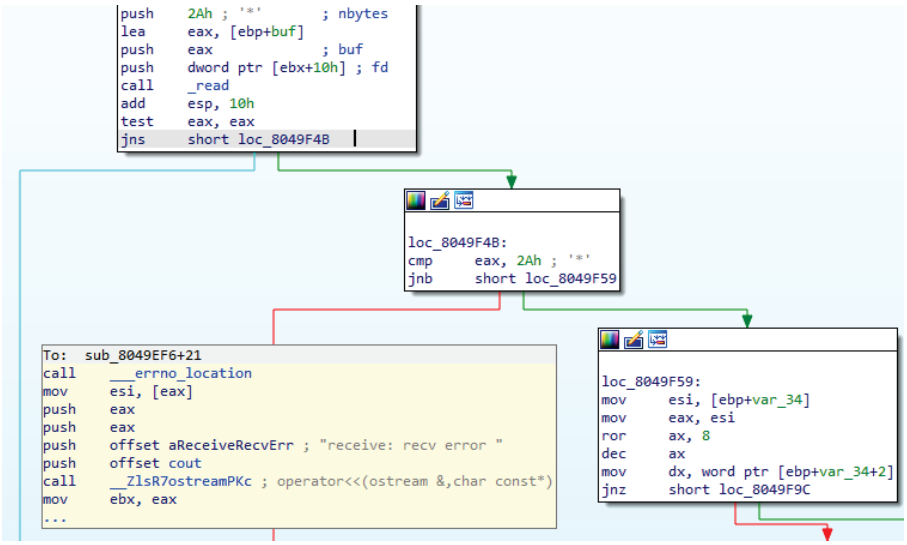


Рисунок 45 — Вероятность перехода в программной функции

Время обработки сообщений в программной процедуре зависит от скорости процессора, скорости ввода/вывода, количества инструкций в процедуре, типов используемых инструкций, размера сообщений. Оценка затраченного времени на выполнение инструкций в процедуре рассчитывается по формуле (73).

$$t = \sum_{i=1}^{n} (IC_i * CPI_i) * t_c \,, \tag{73}$$

где IC_i — количество i-ых инструкций; CPI_i — среднее число тактов, затраченное на выполнение i-ой инструкции; t_c — время одного такта.

Также необходимо учитывать время на работу с внешними шинами данных. Для этого требуется знать частоту и разрядность. Отсюда получается, что время обработки одного сообщения в состоянии S_i есть сумма времени выполнения инструкций в процедуре и время, затраченное на работу с шинами ввода/вывода (74).

$$t = \sum_{i=1}^{n} (IC_i * CPI_i) * t_c + \sum_{j=1}^{k} t_j \,, \tag{74}$$

где t_j — время на обработку i-ого системного прерывания.

Модели СМО программных компонент позволяют не только оценить пропускную способность вычислительных ресурсов, но и выявлять ошибки, связанные с производительностью системы (бесконечные циклы, медленные ресурсы, узкие каналы обмена данными, состояния гонки, запрос на необоснованно большие вычислительные ресурсы и др.). Например, в модели, представленной на Рисунке 29, цветом выделены три состояния S_9, S_{13}, S_{17}, которые теоретически приводят к бесконечному циклу. Похожие ситуации требуют локализации участков ассемблерного кода и дополнительной проверки. Если вероятность наступления бесконечного цикла существует, то такая ошибка должна быть учтена в расчётах производительности.

Объём затрачиваемой памяти при расчётах в дискретные моменты времени определяется в исполняемом коде посредством анализа набора инструкций по выделению и освобождению памяти. При расчётах общего объёма затрачиваемой

памяти достаточно провести подсчёт выделяемого ресурса по ассемблерному коду в соответствии с графами потока управления.

Представленный в разделе 3.1 поиск разнородных ошибок в компонентах системы обеспечивает расчётные модели оценки качества КЭ СУ только количеством этих ошибок. Для расчётов по предлагаемой аналитической модели информации о количестве и типе ошибок недостаточно, необходимо знать степень влияния ошибки на характеристики качества. Анализ исполняемого кода с последующим моделированием программных компонент обеспечивает достаточной информацией для того, чтобы разработать метод оценки влияния ошибки в ПО на характеристики качества КЭ СУ.

3.2 Методика оценки влияния найденных ошибок в ПО на характеристики качества с использованием аппарата нечёткой логики

Ошибки в программных и аппаратных компонентах влияют на значения показателей качества. В процессах оценки качества программно-аппаратных систем требуется применять методы поиска, описания и оценки влияния ошибок на систему и внешнюю среду.

В теории надёжности программных систем используют вероятностные критерии: безотказность, устойчивость к ошибкам, восстанавливаемость. Так как точно определить полное количество ошибок со всеми свойствами прямыми методами измерения невозможно, то используют вероятность нахождения ошибки на определённом участке кода. С помощью вероятностных методов определяют влияние ошибок на критерии надежности.

В области безопасности вычислительных систем используют статистические модели, описывающие оценку влияния по таким критериям, как конфиденциальность, доступность и целостность информации. Стандартом в этой области стала методика расчета Common Vulnerability Scoring System (CVSS) [95]. В отличие от вероятностных моделей, CVSS обладает понятной структурой

расчётов, включает в себя возможность по расширению и полностью детерминирована в общем случае. Существенным недостатком является отсутствие статистических данных, по которым определены формулы для расчётов. Существующие формулы подходят для оценки влияния на безопасность систем общего назначения, но не подходят для специализированных программно-аппаратных систем, что подтверждается критикой CVSS [96].

Таким образом, в условиях отсутствия статистических данных о приоритизации устранения ошибок в программном коде требуется разработать метод и алгоритм расчёта оценки влияния ошибки на характеристики качества.

3.2.1 Анализ методики CVSS при оценке влияния программной ошибки на систему

Методика CVSS версии 3.1 состоит из 8 базовых, 3 временных и 11 контекстных метрик [95]. Оценки делятся на 4 уровня в соответствии со шкалой от 0 до 10: низкий («low») (0.1–3.9), средний («medium») (4.0–6.9), высокий («high») (7.0–8.9) и критический («critical») (9.0–10.0).

Оценка по базовым метрикам основана на расчёте трех показателей: влияние на безопасность информации (E_{iss}) (75), сложность эксплуатации ошибки с целью нарушения безопасности информации (E_e) (76) и влияние на другие компоненты системы (S).

$$E_{iss} = 1 - [(1 - c) * (1 - i) * (1 - a)], \quad (75)$$

где C — конфиденциальность, I — целостность, A — доступность.

$$E_e = 8.22 * av * ac * pr * ui, \quad (76)$$

где av — вектор атаки, ac — сложность атаки, pr — уровень привилегий, ui — взаимодействие с пользователем.

Влияние ошибки на другие компоненты системы учитывается в (77)

$$E_i = \begin{cases} 6.42 * E_{iss}, \text{если } S = 0 \\ 7.52 * (E_{iss} - 0.029) - 3.25 * (E_{iss} - 0.02)^{15}, \end{cases} \quad (77)$$

где S — оценка влияния на другие компоненты системы.

Оценка влияния ошибки на информационную безопасность (78) имеет три состояния:

$$E_{bs} = \begin{cases} 0, \text{если } E_i \leq 0 \\ \min\left((E_i + E_e)\ ,10\right), \text{если } S = 0 \\ \min(1.08 * (E_i + E_e)\ ,10), \text{если } S = 1 \end{cases} \quad (78)$$

На Рисунке 41 изображены результаты расчёта оценок E_{bs} (Base Core), E_{iss} (ISS), E_e (Exploitable), S (Scope), E_i (Impact). Из графика можно установить полную корреляцию между E_{bs} и E_e. Также формула (78) и график (Рисунок 46) подтверждают, что уровень E_{bs} зависит от показателя E_i.

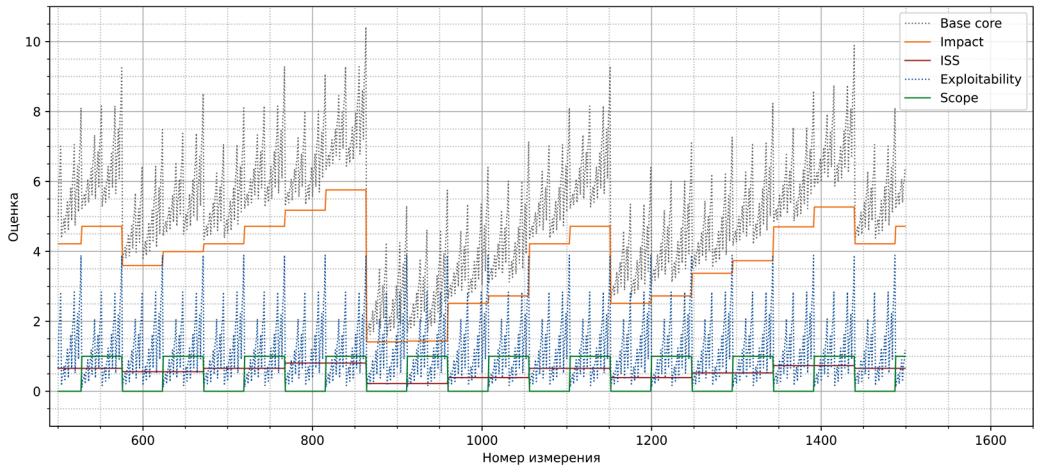


Рисунок 46 — График оценок CVSS

Если обратиться к формулам 76–78, то возникает вопрос: каким образом получены значения коэффициентов показателей? На этот вопрос отвечает техническая документация, представленная в историческом обзоре создания методики CVSS [97]. Как следует из письма Джеймса Х. Йена (кандидата наук, сотрудника Национального института стандартов и технологий) к группе разработчиков CVSS, коэффициенты для формул 75–78 подбирались на основе статистического анализа и уточнялись от версии 1.0 до 3.1. Весовые коэффициенты подбирались в рамках расчёта простого среднего взвешенного значения оценки. Отсюда следует, что в момент необходимости модификации методики CVSS и отсутствия статистических данных об ошибках изменить модель расчётов оценки представляется трудоёмкой задачей. Предлагается разрешить описанную

проблему, заменив существующий математический аппарат на аппарат нечёткой логики [98].

3.2.2 Использование нечёткой логики в оценке влияния программной ошибки на систему

Математическая теория нечёткой логики вводит такое понятие, как «лингвистическая переменная», значения которой (термы) определяются через нечёткие множества. Нечётким множеством A называется такая совокупность упорядоченных пар $\{(x_i, \mu_A(x_i)) \mid x_i \in X\}$, где x_i — элемент универсального множества X, а $\mu_A(x_i)$ — функция принадлежности, которая определяет степень принадлежности элемента x_i нечёткому множеству. Система нечёткого логического вывода включает входные и выходные лингвистические переменные, а также базу правил, по которой будет производиться расчёт целевого показателя.

Методика расчёта CVSS имеет 8 базовых метрик, которые имеют качественное и количественное выражение. Расчёт базовой оценки происходит по формулам 57–60 путём подстановки количественных значений метрик. Для создания системы нечёткого логического вывода в качестве лингвистических переменных использованы базовые метрики, а качественное выражение каждого из них определяют термы. При создании функций принадлежности для каждого терма использовались числовые выражения базовых метрик. В Таблице 10 представлены входные лингвистические переменные с формулами функций принадлежности.

Таблица 10 — Параметры входных переменных

Входная переменная	Входной терм	Тип функции принадлежности	Формула функции принадлежности
Вектор атаки (AV)	Сетевой (network)	Z-функция	$f(x,0.25,0.3)\begin{cases}1, x<0.25\\ \frac{0.3-x}{0.25}, 0.25\le x\le 0.3\\ 0, x>0.3\end{cases}$
	Смежная сеть (adjacent)	Треугольная	$f(x,0.2,0.55,0.6)\begin{cases}0.2, x<0\\ \frac{x-0.2}{0.35}, 0.2\le x\le 0.55\\ \frac{0.6-x}{0.05}, 0.55<x\le 0.6\\ 0, x>0.6\end{cases}$
	Локальный (local)	Трапециевидная	$f(x,0.5,0.61,0.7,0.8)\begin{cases}0, x<0.5\\ \frac{x-0.5}{0.11}, 0.5\le x\le 0.6\\ 1, 0.61<x\le 0.7\\ \frac{0.8-x}{0.1}, 0.7<x\le 0.8\\ 0, x>0.8\end{cases}$
	Физический (physic)	S-функция	$f(x,0.7,0.8)\begin{cases}0, x<0.7\\ \frac{x-0.7}{0.1}, 0.7\le x\le 0.8\\ 1, x>0.8\end{cases}$
Сложность атаки (AC)	Высокая (high)	Трапециевидная	$f(x,0,0.3,0.5,0.7)\begin{cases}0, x<0\\ \frac{x}{0.3}, 0\le x\le 0.3\\ 1, 0.3<x\le 0.5\\ \frac{0.7-x}{0.2}, 0.5<x\le 0.7\\ 0, x>0.7\end{cases}$
	Низкая (low)	S-функция	$f(x,0.6,0.7)\begin{cases}0, x<0.6\\ \frac{x-0.6}{0.1}, 0.6\le x\le 0.7\\ 1, x>0.7\end{cases}$
Уровень привилегий (PR)	Высокий (high)	Z-функция	$f(x,0.3,0.4)\begin{cases}1, x<0.3\\ \frac{0.4-x}{0.1}, 0.3\le x\le 0.4\\ 0, x>0.4\end{cases}$
	Низкий (low)	Треугольная	$f(x,0.2,0.62,0.8)\begin{cases}0.2, x<0\\ \frac{x-0.2}{0.42}, 0.2\le x\le 0.62\\ \frac{0.8-x}{0.18}, 0.62<x\le 0.8\\ 0, x>0.8\end{cases}$
	Не требуется (none)	S-функция	$f(x,0.7,0.8)\begin{cases}0, x<0.7\\ \frac{x-0.7}{0.1}, 0.7\le x\le 0.8\\ 1, x>0.8\end{cases}$
Взаимодействие с пользователем (UI)	Требуется (required)	Z-функция	$f(x,0.65,0.7)\begin{cases}1, x<0.65\\ \frac{0.7-x}{0.05}, 0.65\le x\le 0.7\\ 0, x>0.7\end{cases}$
	Не требуется (none)	S-функция	$f(x,0.65,0.8)\begin{cases}0, x<0.65\\ \frac{x-0.65}{0.15}, 0.65\le x\le 0.8\\ 1, x>0.8\end{cases}$

Продолжение таблицы 10

Входная переменная	Входной терм	Тип функции принадлежности	Формула функции принадлежности
Влияние на целостность (C) Влияние на конфиденциальность (I) Влияние на доступность (A)	Не оказывает (not)	Z-функция	$f(x,0.1,0.2)\begin{cases}1, x<0.1\\ \frac{0.2-x}{0.1}, 0.1\le x\le 0.2\\ 0, x>0.2\end{cases}$
	Низкое (low)	Трапециевидная	$f(x,0.1,0.2,0.35,0.45)=\begin{cases}0, x<0.1\\ \frac{x-0.1}{0.1}, 0.1\le x\le 0.2\\ 1, 0.2<x\le 0.35\\ \frac{0.45-x}{0.1}, 0.35<x\le 0.45\\ 0, x>0.45\end{cases}$
	Высокое (high)	S-функция	$f(x,0.4,0.56)\begin{cases}0, x<0.4\\ \frac{x-0.4}{0.16}, 0.4\le x\le 0.56\\ 1, x>0.56\end{cases}$
Связь с другими компонентами системы (S)	Отсутствует (unchanged)	Z-функция	$f(x,0.5,0.6)\begin{cases}1, x<0.5\\ \frac{0.5-x}{0.1}, 0.5\le x\le 0.6\\ 0, x>0.6\end{cases}$
	Существует (changed)	S-функция	$f(x,0.5,0.7)\begin{cases}0, x<0.5\\ \frac{x-0.5}{0.2}, 0.5\le x\le 0.7\\ 1, x>0.7\end{cases}$
Сложность атаки (AC)	Высокая (high)	Трапециевидная	$f(x,0,0.3,0.5,0.7)=\begin{cases}0, x<0\\ \frac{x}{0.3}, 0\le x\le 0.3\\ 1, 0.3<x\le 0.5\\ \frac{0.7-x}{0.2}, 0.5<x\le 0.7\\ 0, x>0.7\end{cases}$
	Низкая (low)	S-функция	$f(x,0.6,0.7)\begin{cases}0, x<0.6\\ \frac{x-0.6}{0.1}, 0.6\le x\le 0.7\\ 1, x>0.7\end{cases}$
Уровень привилегий (PR)	Высокий (high)	Z-функция	$f(x,0.3,0.4)\begin{cases}1, x<0.3\\ \frac{0.4-x}{0.1}, 0.3\le x\le 0.4\\ 0, x>0.4\end{cases}$
	Низкий (low)	Треугольная	$f(x,0.2,0.62,0.8)\begin{cases}0.2, x<0\\ \frac{x-0.2}{0.42}, 0.2\le x\le 0.62\\ \frac{0.8-x}{0.18}, 0.62<x\le 0.8\\ 0, x>0.8\end{cases}$
	Не требуется (none)	S-функция	$f(x,0.7,0.8)\begin{cases}0, x<0.7\\ \frac{x-0.7}{0.1}, 0.7\le x\le 0.8\\ 1, x>0.8\end{cases}$
Взаимодействие с пользователем (UI)	Требуется (required)	Z-функция	$f(x,0.65,0.7)\begin{cases}1, x<0.65\\ \frac{0.7-x}{0.05}, 0.65\le x\le 0.7\\ 0, x>0.7\end{cases}$

Продолжение таблицы 10

Входная переменная	Входной терм	Тип функции принадлежности	Формула функции принадлежности
	Не требуется (none)	S-функция	$f(x, 0.65, 0.8)\begin{cases} 0, x < 0.65 \\ \frac{x - 0.65}{0.15}, 0.65 \le x \le 0.8 \\ 1, x > 0.8 \end{cases}$
Влияние на целостность (C) Влияние на конфиденциальность (I) Влияние на доступность (A)	Не оказывает (not)	Z-функция	$f(x, 0.1,0.2)\begin{cases} 1, x < 0.1 \\ \frac{0.2 - x}{0.1}, 0.1 \le x \le 0.2 \\ 0, x > 0.2 \end{cases}$
	Низкое (low)	Трапециевидная	$f(x, 0.1, 0.2, 0.35, 0.45) = \begin{cases} 0, x < 0.1 \\ \frac{x - 0.1}{0.1}, 0.1 \le x \le 0.2 \\ 1, 0.2 < x \le 0.35 \\ \frac{0.45 - x}{0.1}, 0.35 < x \le 0.45 \\ 0, x > 0.45 \end{cases}$
	Высокое (high)	S-функция	$f(x, 0.4, 0.56)\begin{cases} 0, x < 0.4 \\ \frac{x - 0.4}{0.16}, 0.4 \le x \le 0.56 \\ 1, x > 0.56 \end{cases}$
Связь с другими компонентами системы (S)	Отсутствует (unchanged)	Z-функция	$f(x, 0.5,0.6)\begin{cases} 1, x < 0.5 \\ \frac{0.5 - x}{0.1}, 0.5 \le x \le 0.6 \\ 0, x > 0.6 \end{cases}$
Связь с другими компонентами системы (S)	Существует (changed)	S-функция	$f(x, 0.5, 0.7)\begin{cases} 0, x < 0.5 \\ \frac{x - 0.5}{0.2}, 0.5 \le x \le 0.7 \\ 1, x > 0.7 \end{cases}$

В Таблице 11 представлены термы нечёткой оценки влияния на безопасность с формулами функций принадлежности.

Таблица 11 — Параметры выходной переменной

Выходной терм	Тип функции принадлежности	Формула функции принадлежности
Не влияет (none)	Z-функция	$f(x, 0.1,0.2)\begin{cases} 1, x < 0.1 \\ \frac{0.2 - x}{0.1}, 0.1 \le x \le 0.2 \\ 0, x > 0.2 \end{cases}$
Низкое влияние (low)	Треугольная	$f(x, 0.1, 0.25, 0.4)\begin{cases} 0, x < 0.1 \\ \frac{x - 0.1}{0.15}, 0.1 \le x \le 0.25 \\ \frac{0.4 - x}{0.15}, 0.25 < x \le 0.4 \\ 0, x > 0.4 \end{cases}$
Среднее влияние (medium)	Треугольная	$f(x, 0.3, 0.55, 0.8)\begin{cases} 0, x < 0.3 \\ \frac{x - 0.3}{0.25}, 0.3 \le x \le 0.55 \\ \frac{0.8 - x}{0.25}, 0.55 < x \le 0.8 \\ 0, x > 0.8 \end{cases}$

Продолжение таблицы 11

Выходной терм	Тип функции принадлежности	Формула функции принадлежности
Высокое влияние (high)	Треугольная	$f(x, 0.6, 0.75, 0.95)\begin{cases} 0, x < 0.6 \\ \frac{x - 0.6}{0.15}, 0.6 \le x \le 0.75 \\ \frac{0.95 - x}{0.2}, 0.75 < x \le 0.95 \\ 0, x > 0.95 \end{cases}$
Влияние критично (critical)	S-функция	$f(x, 0.8, 0.9)\begin{cases} 0, x < 0.8 \\ \frac{x - 0.8}{0.1}, 0.1 \le x \le 0.9 \\ 1, x > 0.9 \end{cases}$

Если создавать базу нечётких правил по всем входным переменным, то без оптимизации получается 2592 правила, что сократит возможности метода при модификации в специальных случаях, а также увеличит время расчёта оценки.

Решение сформулированной проблемы лежит в реализации методики CVSS. По аналогии, целесообразно разбить расчёт оценки на три следующих показателя: нечёткая оценка влияния на безопасность информации (FE_{iss}), нечёткая оценка сложности эксплуатации ошибки с целью нарушения безопасности информации (FE_e) и нечёткая оценка влияния ошибки на компоненты системы (FE_i). Расчёт каждого показателя основан на создании выходных термов, функций принадлежности и базы нечётких правил, которые в свою очередь являются входными переменными для расчёта нечёткой оценки влияния ошибки на систему FE_{bs}. В Таблице 12 представлены количественные показатели математического метода расчета FE_{bs}.

Таблица 12 — Количественные характеристики метода

Название показателя	Количество входных переменных	Количество выходных термов	Количество нечётких правил	Метод дефазификации
Нечёткая оценка влияния на безопасность информации (FE_{iss})	3	5	27	Относительно центра области $x_c = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)}$

Продолжение таблицы 12

Название показателя	Количество входных переменных	Количество выходных термов	Количество нечётких правил	Метод дефазификации
Нечёткая оценка сложности эксплуатации ошибки с целью нарушения безопасности информации (FE_e)	4	6	48	Выбор минимального из максимальных значений
Нечёткая оценка влияния ошибки на компоненты системы (FE_i)	2	5	10	Выбор максимального из максимальных значений
Нечёткая оценка влияния ошибки на систему (FE_{bs})	3	5	36	Выбор минимального из максимальных значений

С точки зрения принятия решений для обеспечения безопасности информации, требуется оценить влияние ошибки на систему. Решение принимается на основе определённого уровня влияния ошибки. Поэтому определение конкретного уровня является целью расчётов оценки влияния программной ошибки на систему. На Рисунке 47 (отмечено зелёным цветом) показано, как числовой показатель базовой оценки CVSS заполняет интервалы, которые соответствуют разным уровням — от низкого до критического. В отличие от методики CVSS, в результате применения аппарата нечёткой логики получаем значение уровня ошибки без числового показателя (на Рисунке 47 отмечено чёрным цветом). Таким образом, принятие решений на основе расчётов базовой оценки с помощью аппарата нечёткой логики обосновано, так как достигаются те же цели, что и при расчётах CVSS.

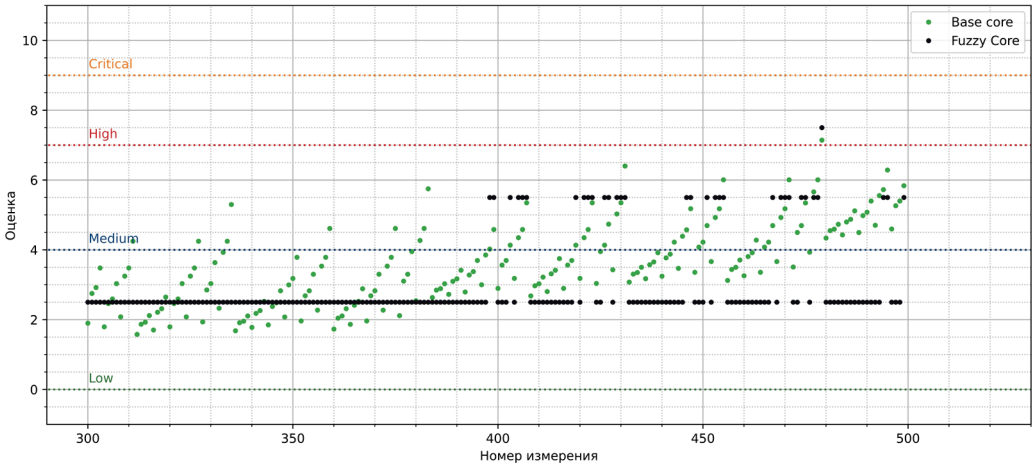


Рисунок 47 — График расчётов оценки CVSS и нечёткой оценки влияния

Вся логика математического аппарата расчёта оценки влияния программной ошибки на систему запрограммирована на языке Python с использованием библиотеки skfuzzy. В отдельные функции вынесены расчёты оценок FE_{iss}, FE_e, FE_i и FE_{bs}.

Представленная методика имеет гибкую структуру, которая может быть модифицирована под общие или специальные требования. В отличие от CVSS, где входные значения определены на основе статистического анализа, предлагаемая методика позволяет изменять числовые значения входных термов, что упрощает настройку оценки под конкретные КЭ СУ. База нечётких правил может быть расширена специальными показателями, которые необходимо учитывать при расчётах.

3.2.3 Разработка метрик оценки влияния ошибки в программных компонентах на характеристики качества КЭ СУ

Расчёты степени влияния ошибки в программных компонентах на базовые характеристики качества требуют введения новых метрик и значений. По аналогии с безопасностью, разработаем метрики для функциональной пригодности, надёжности и производительности. Разделим метрики по трём критериям: 1-ая группа метрик – «Сложность воспроизведения ошибки», 2-я группа метрик –

«Влияние на другие компоненты», 3-я группа метрик – «Влияние на показатели отдельной характеристики качества». Так как, согласно разработанному методу, метрики являются входными лингвистическими переменными, то значения этих метрик являются входными термами, которым соответствует своя функция принадлежности. В Таблице 13 представлены метрики с описанием и возможные значения входных термов для расчёта оценок влияния.

Таблица 13 — Метрики оценки влияния ошибки на систему

Характеристика	Метрика	Значения	Описание
1-я группа метрик			
Безопасность	Вектор атаки	Сетевой Смежная сеть Локальный Физический	Указывает на расположение источника возможной угрозы безопасности
	Сложность атаки	Высокая Низкая	Комплексная оценка сложности эксплуатации ошибки злоумышленником
	Уровень привилегий	Высокий Низкий Не требуется	Выбирается максимально низкий уровень привилегий, которым должен обладать злоумышленник, чтобы эксплуатировать ошибку
	Взаимодействие с пользователем	Требуется Не требуется	Значение зависит от участия пользователя системы при эксплуатации ошибки
Функциональная пригодность	Компонент системы	Сетевой Программный Аппаратный	Указывает на подсистему, где обнаружена ошибка
	Сложность алгоритма нарушения функциональности	Высокая Низкая	Оценка сложности пути до кода, содержащего ошибку. Оценка состоит из вероятности формирования данных, обработка которых приводит к ошибке, и вероятности попадания потока управления на код, содержащий ошибку.
	Уровень привилегий	Высокий Низкий Не требуется	Выбирается тот уровень привилегий, которым должен обладать пользователь, чтобы проявилась ошибка
	Взаимодействие с пользователем	Требуется Не требуется	Значение зависит от участия пользователя системы для проявления ошибки
Производительность	Источник нагрузки	Сетевой Смежная сеть Локальный Физический	Указывает на расположение источника данных, при обработке которых возникает ошибка, влияющая на производительность системы
	Сложность алгоритма возникновения ошибки	Высокая Низкая	Оценка сложности алгоритма, который приводит к выполнению ошибочного кода.
	Уровень привилегий	Высокий Низкий Не требуется	Выбирается тот уровень привилегий, которым должен обладать пользователь, чтобы проявилась ошибка

Продолжение таблицы 13

Характеристика	Метрика	Значения	Описание
Производительность	Взаимодействие с пользователем	Требуется Не требуется	Значение зависит от участия пользователя системы для проявления ошибки
Надёжность	Источник данных отказа	Сетевой Смежная сеть Локальный Физический	Указывает на расположение источника данных, при обработке которых возникает ошибка, вызывающая сбой или отказ в системе
	Сложность алгоритма возникновения отказа	Высокая Низкая	Оценка сложности возникновения отказа из-за ошибки
	Уровень привилегий	Высокий Низкий Не требуется	Выбирается тот уровень привилегий, которым должен обладать пользователь, чтобы сформировать данные, которые вызовут отказ или сбой в системе из-за ошибки
	Взаимодействие с пользователем	Требуется Не требуется	Значение зависит от участия пользователя системы для вызова отказа или сбоя в системе из-за ошибки
2-я группа метрик			
Безопасность	Влияние на другие компоненты системы	Не оказывает Оказывает	Принимается значение «1» или «0» соответственно в зависимости от наличия или отсутствия влияния ошибки на смежные компоненты системы
Функциональная пригодность	Влияние на другие компоненты системы	Не оказывает Оказывает	
Производительность	Влияние на другие компоненты системы	Не оказывает Оказывает	
Надёжность	Влияние на другие компоненты системы	Не оказывает Оказывает	
3-я группа метрик			
Безопасность	Влияние на конфиденциальность	Не оказывает Низкое Высокое	Определяет средневзвешенное значение вероятности влияния ошибки на такие ПК безопасности, как конфиденциальность, целостность и доступность информации
	Влияние на целостность		
	Влияние на доступность		
Функциональная пригодность	Влияние на полноту	Не оказывает Низкое Высокое	Определяет средневзвешенное значение вероятности влияния ошибки на такие ПК функциональности, как полнота реализации, корректность и точность выполнения программно реализованных алгоритмов
	Влияние на корректность		
	Влияние на точность		

Продолжение таблицы 13

Характеристика	Метрика	Значения	Описание
Производительность	Влияние на пропускную способность	Не оказывает Низкое Высокое	Определяет средневзвешенное значение вероятности влияния ошибки на такие ПК производительности, как пропускная способность канала обработки данных, время отклика на обработку заявки и объём вычислительных ресурсов
	Влияние на время отклика		
	Влияние на объём ресурсов		
Надёжность	Влияние на безотказность	Не оказывает Низкое Высокое	Определяет средневзвешенное значение вероятности влияния ошибки на такие ПК надёжности, как вероятность безотказной работы, среднее время на восстановление после отказа, вероятность отказа в обслуживании
	Влияние на восстанавливаемость		
	Влияние на доступность		

После того как метрики введены, необходимо разработать способ определения их значений. Так как метод оценки влияния ошибки в программных компонентах является частью предлагаемого алгоритма поиска ошибок, то на вход метода поступает информация, полученная благодаря анализу исполняемого кода. За счёт этого эксперт имеет полный доступ к логике работы программных компонент, к графу потока управления, к вычислительным операциям, к графу потока данных (Рисунок 48).

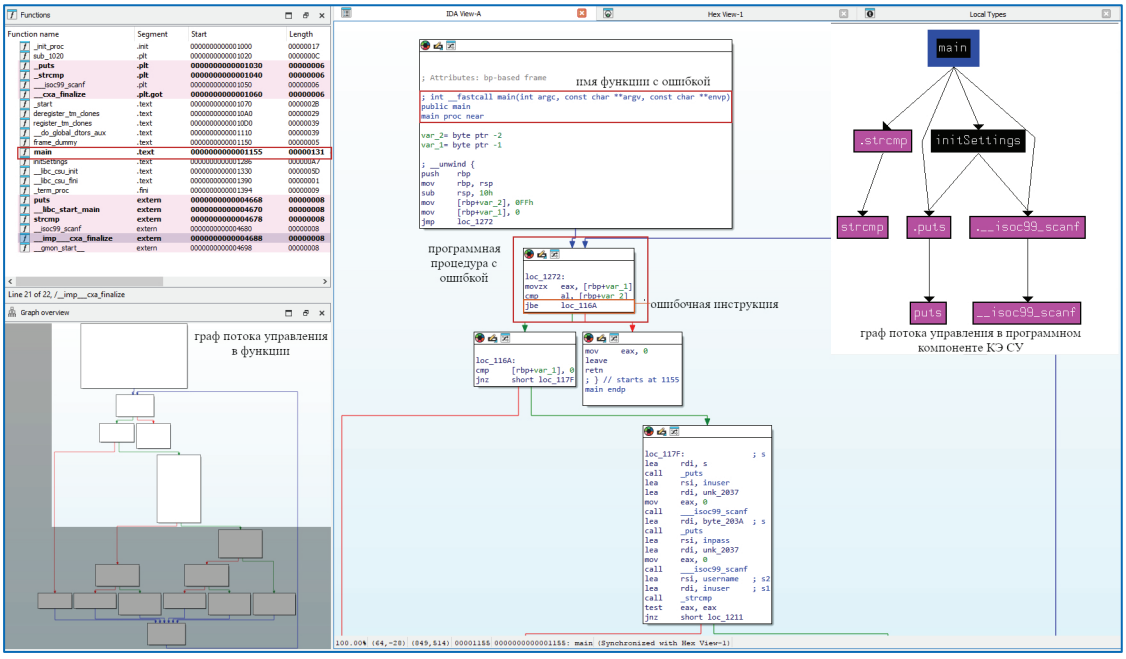


Рисунок 48 — Информация об ошибке в программном компоненте КЭ СУ

Например, для всех характеристик применяется метрика «уровень привилегий». При обнаружении ошибки эксперт понимает следующее:

1. Ошибка находится в программной функции с именем *main*. Это первая пользовательская функция в программе.
2. Ошибочная программная процедура выполняется всегда без условий.
3. Ограничения на ввод данных отсутствуют. Достаточно ввести любые авторизационные данные больше 255 раз, что вызовет повторную инициализацию устройства.
4. Ошибочная функция вызывается первой согласно трассе выполнения, также это наблюдается по графу потока управления в программном компоненте КЭ СУ.

На основе полученной информации устанавливается, что пользователь с минимальным доступом к системе (значение метрики «уровень привилегий» соответствует «не требуются») способен вызвать ошибку в системе.

Отсюда следует, что использование информации, полученной с помощью анализа исполняемого кода, обеспечивает метод оценки влияния ошибки в программном обеспечении на характеристики качества КЭ СУ достаточной полнотой.

3.3 Комплексирование аналитической модели, алгоритма поиска ошибок и частного метода в метод оценки качества КЭ СУ

Логическое объединение предложенного алгоритма поиска ошибок, модели оценки качества КЭ СУ, метода оценки влияния программной ошибки на КЭ СУ, методов анализа исполняемого кода КЭ СУ и существующего метода модельно-ориентированного системного инжиниринга позволяет построить метод оценки качества КЭ СУ при переносе ПО на ААП.

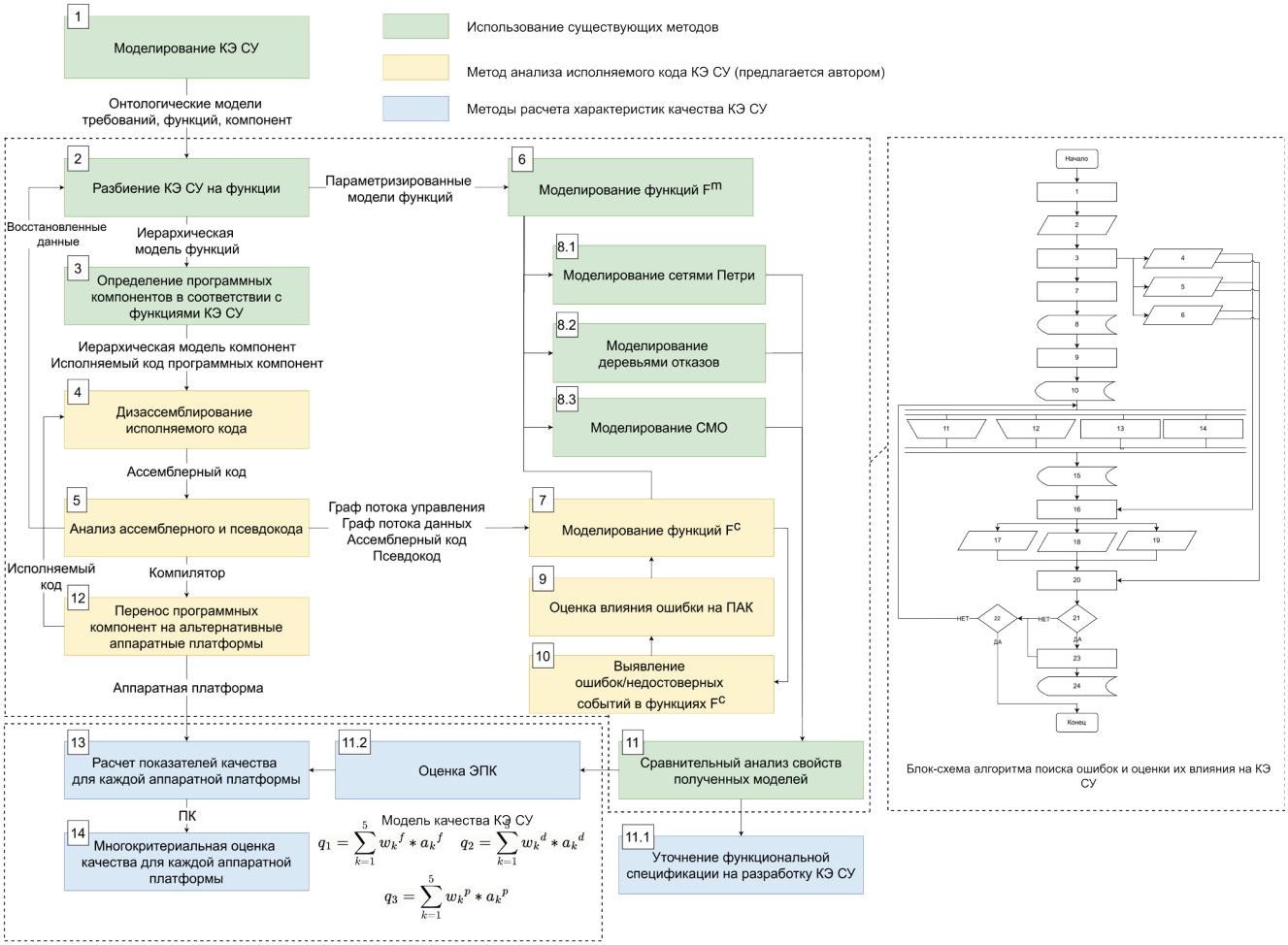


Рисунок 49 — Схема метода оценки характеристик качества (функциональная пригодность, надёжность, производительность)

На схеме (Рисунок 49) блоки 1, 2, 3, 6, 7, 8.1, 8.2, 8.3 объединены в метод MBSE, который является опорным при построении метода оценки качества КЭ СУ. Блоки 4, 5, 12, 7 включают анализ исполняемого кода, который основан на статическом и динамическом анализе исполняемого кода, анализе графов потока данных и управления, анализе трасс вызова функций, анализе структур и типов данных, восстановлении математических операций, восстановлении логики работы программных компонент, восстановлении технической документации на программные компоненты, восстановлении исходных кодов программных компонент. Блоки 10 и 11 отвечают за выявление ошибок в программных компонентах, результаты выполнения этой части метода являются входными данными для частного метода оценки влияния ошибки в ПО КЭ СУ (блок 9). В результате сравнительного анализа априорных и получаемых моделей функций

системы целесообразно проводить уточнение функциональной спецификации на разработку КЭ СУ (блок 11.1). Для расчёта многокритериальной оценки качества КЭ СУ используется разработанная аналитическая модель оценки качества (блоки 11.2, 13, 14).

Составные части метода, характеризующие алгоритм (блоки 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), частный метод (блок 9) и аналитическую модель (блоки 11.2, 13, 14), расположены не последовательно, а имеют сложные связи. Такая организация метода продиктована итерационным подходом, что реализовано с помощью параллельных и обратных связей.

Повышение точности в разработанном методе достигается за счёт моделирования программных компонент КЭ СУ каждый раз при переносе ПО на ААП и проведения сравнительного анализа между всеми моделями в рамках одного класса с целью выявления программных и аппаратно-зависимых ошибок.

Метод применим в случаях ограниченных входных данных: отсутствие исходных кодов на программные компоненты, отсутствие технической документации. С определёнными допущениями (наличие аппаратной спецификации на образец, наличие исполняемого кода) метод применим, когда отсутствует образец системы.

Таким образом, разработан метод оценки качества компьютерных элементов системы управления, обеспечивающий высокий уровень точности оценки для управления модернизацией с использованием переноса программного обеспечения на альтернативные аппаратные платформы.

3.4 Выводы

Разработан алгоритм поиска ошибок в исполняемом коде, который основан на моделировании отдельных программных компонент за счёт существующих

методов обратного проектирования, что позволяет обнаружить ошибки в готовом образце системы без технической документации и исходных кодов ПО.

Так как разработанная аналитическая модель качества КЭ СУ требует качественных метрик (для учёта найденных ошибок), то разработана методика оценки влияния программной ошибки на характеристики качества КЭ СУ, которая, в отличие от существующих, использует аппарат нечёткой логики, что позволяет ей решить проблему отсутствия статистических данных.

Комплексирование всех результатов исследования позволяет сформулировать метод оценки качества КЭ СУ, основанный на анализе моделей исполняемого кода и алгоритме поиска программных ошибок. Метод способен выявлять аппаратно-зависимые ошибки, возникающие в исполняемом коде в момент переноса программного обеспечения на альтернативные аппаратные платформы, учитывать их влияние на базовые характеристики качества, что обеспечивает расчёты многокритериальной оценки качества с требуемым уровнем точности. Результат позволяет усовершенствовать поиск ошибок в перенесённом программном коде в части оценки и прогнозирования качества при модернизации компьютерных элементов системы управления.

ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНАЯ ПРОВЕРКА МОДЕЛИ И МЕТОДА ОЦЕНКИ КАЧЕСТВА КЭ СУ. РЕКОМЕНДАЦИИ ПО УПРАВЛЕНИЮ МОДЕРНИЗАЦИЕЙ КЭ СУ С ИСПОЛЬЗОВАНИЕМ ПЕРЕНОСА ПО НА ААП

Проведение экспериментальной проверки разработанных модели и метода сопровождается комплексом технических мероприятий, которые направлены на подтверждение достижения поставленной цели. Мероприятия обеспечиваются набором входных данных, применением существующих или разработанных инструментов, а также перечнем выходных данных. Получаемая информация структурируется и заносится в базу данных. На Рисунке 50 представлена схема проведения экспериментальной проверки с указанием используемых инструментов, проводимых мероприятий и перечнем входных и выходных данных.

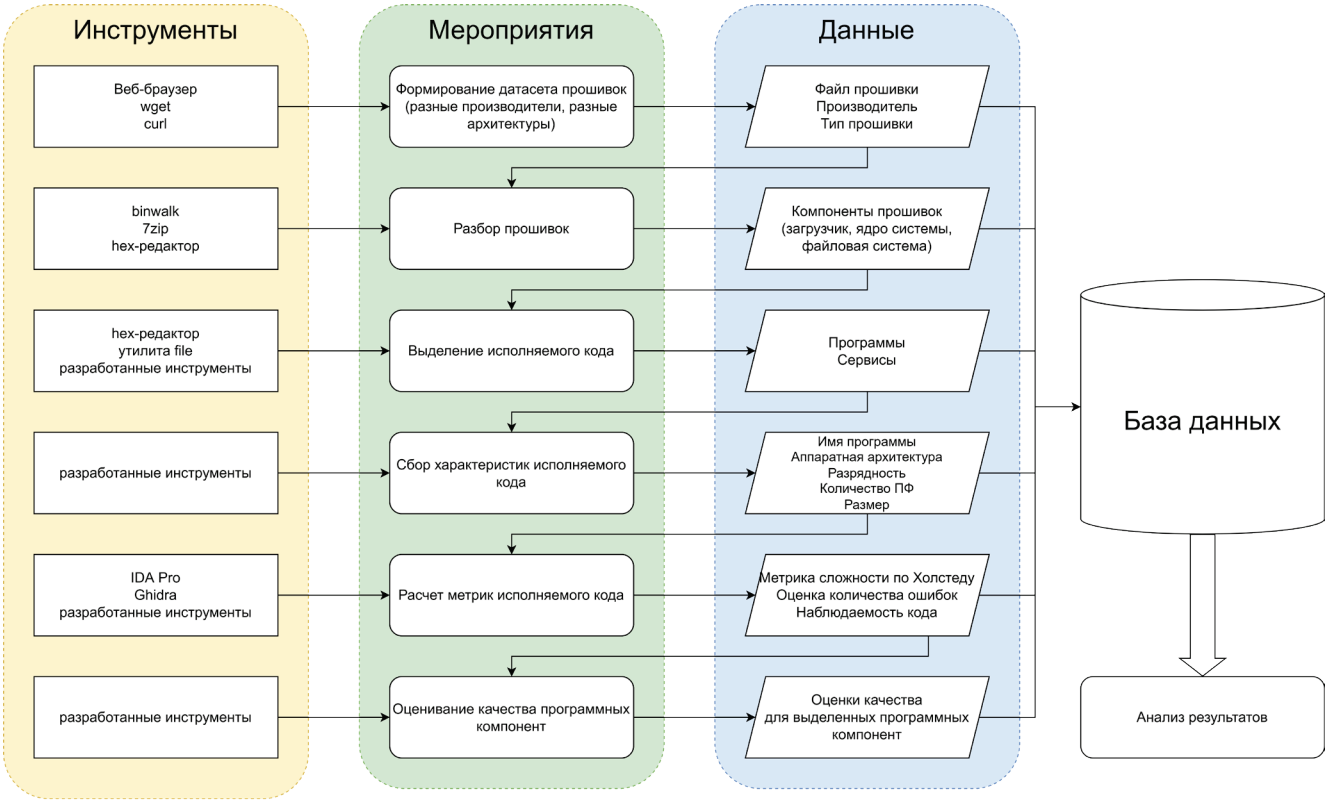


Рисунок 50 — Схема экспериментальной проверки результатов исследования

Эксперимент состоит из трёх этапов:

1. Сбор данных и разработка инструментов анализа информации на каждом шаге проведения эксперимента.

2. Проверка утверждения о том, что предлагаемая аналитическая модель адекватно позволяет рассчитывать базовые характеристики качества КЭ СУ.

3. Проверка утверждения о том, что результаты применения разработанного метода оценки качества при модернизации системы управления путём переноса ПО на ААП обладают точностью методов, используемых при разработке системы.

Так как в основе технической реализации каждой сложной системы управления находятся устройства приёма/передачи информации, то в качестве экспериментального образца (ЭО) КЭ СУ выступает телекоммуникационное оборудование (ТКО) различных производителей. На выбор в пользу ТКО как ЭО влияют такие факторы, как:

— встроенное программное обеспечение (ВсПО) ТКО обладает большим объёмом разнородного исполняемого кода;

— производители ТКО используют общую кодовую базу для устройств на разных аппаратных платформах, что является переносом ПО на ААП;

— исходные данные для проведения эксперимента содержатся в открытом доступе, что позволяет повторить полученные результаты.

Исходя из представленных факторов выбрано 112 моделей устройств трёх производителей ТКО на базе ОС Linux с аппаратными архитектурами x86_64, arm, armel, arm64, mips, mmips, smips, mips64, PowerPC.

4.1 Разработка инструментов сбора, хранения и обработки информации для проведения экспериментальной проверки

Входными данными для проведения экспериментальной проверки результатов исследования являются файл прошивки (образ ВсПО), информация о производителе и типе. Производители ТКО публикуют требуемую информацию в открытом доступе, а с помощью существующих веб-инструментов её возможно сохранить и обработать. Веб-браузеры позволяют собрать данные в ручном режиме, но такой способ требует значительного ресурса времени. Инструменты

wget или *curl* обладают функциями автоматизированной загрузки информации в большом объёме, что не требует разработки новых программ. Таким образом, утилитой *wget* обработаны веб-ресурсы производителей ТКО и получено 224 файла образов ВсПО.

Разбор прошивок предполагает знание о типе прошивки, так как от этого зависит выбор инструмента [99, 100]. Тип прошивки — это множество значений признаков, которые соответствуют исследуемому файлу прошивки. Прошивки классифицируются по следующим признакам:

1. Защита от анализа:
   — шифрованные;
   — кодированные;
   — закрытые (проприетарные) форматы;
   — открытые форматы (защита от анализа отсутствует).
2. Контроль целостности:
   — наличие простых контрольных сумм (CRC, checksum и др.);
   — наличие хеш-сумм (MD5, SHA-1, SHA-512 и др.);
   — наличие электронно-цифровой подписи;
   — контроль целостности отсутствует.
3. Тип операционной системы:
   — открытые ОС (Unix-подобные, VxWorks, QNX и др.);
   — закрытые ОС (WinCE, Cisco IOS и др.).
4. Наличие файловой системы (ФС):
   — упакованная ФС (squashFS, cramFS, UBIFS и др.);
   — журналируемая ФС (ext2/3/4, FAT32 и др.);
   — ФС отсутствует.
5. Структура ОС:
   — ОС с отдельными программными компонентами;
   — все функции ОС в едином исполняемом коде;
   — комбинированная ОС.

Некоторые признаки накладывают существенные ограничения для анализа, поэтому существуют механизмы их быстрого выявления. Так, например, если файл

прошивки зашифрован, то дальнейший анализ сводится к поиску ключей шифрования на готовом образце системы или необходимо делать запрос к производителю. Чтобы отобрать только нешифрованные прошивки, используется энтропийный анализ, который реализован в инструментах автоматического определения [101]. Поэтому исследования ограничены следующими типами прошивок:

1. {открытые форматы; наличие простых контрольных сумм; открытые ОС; упакованная ФС; ОС с отдельными программными компонентами}.
2. {закрытые форматы; наличие хеш-сумм; открытые ОС; упакованная ФС; ОС с отдельными программными компонентами}.
3. {закрытые форматы; наличие электронно-цифровой подписи; открытые ОС; упакованная ФС; ОС с отдельными программными компонентами}.

На следующем шаге проходит разбор файла прошивки на составные компоненты. Набор инструментов *binwalk* способен выделить основные части прошивки (типовые заголовки, загрузчик, ядро ОС, ФС), а дополнительные опции распаковывают ФС (Рисунок 51).

```
$ binwalk -eM /home/user/Firmware/Zyxel/KN-1111_stable_3.02.C.1.0-0_firmware.bin
Scan Time:     2024-09-18 08:39:39
Target File:   /home/user/Firmware/Zyxel/KN-1111_stable_3.02.C.1.0-0_firmware.bin
MD5 Checksum:  48bc877261ad7af145f14103292ffe42
Signatures:    411
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0x4B80F67C, created:
2019-10-14 12:56:50, image size: 1386911 bytes, Data Address: 0x80000000, Entry Point: 0x80319CD8, data CRC:
0x2276449B, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "KN-1111"
64            0x40            LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes,
uncompressed size: 4188620 bytes
1441816       0x160018        PEM certificate
1443097       0x160519        PEM certificate
Scan Time:     2024-09-18 08:39:40
Target File:   /home/user/Firmware/Zyxel/_KN-1111_stable_3.02.C.1.0-0_firmware.bin-0.extracted/40
MD5 Checksum:  634e5737356bd3f79fc696746c2a88f9
Signatures:    411
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
3684512       0x3838A0        CRC32 polynomial table, little endian
3685536       0x383CA0        xz compressed data
3690320       0x384F50        Unix path: /lib/firmware/updates/4.9-ndm-3
3732036       0x38F244        Neighborly text, "NeighborSolicitsp6InMsgs"
3732056       0x38F258        Neighborly text, "NeighborAdvertisementsrs"
3742109       0x39199D        Neighborly text, "neighbor table overflow!is %x"
3796046       0x39EC4E        Neighborly text, "neighbor %.2x%.2x.%pM lost"
4042752       0x3DB000        ELF, 32-bit LSB MIPS64 shared object, MIPS, version 1 (SYSV)
```

Рисунок 51 — Результат работы программы *binwalk*

Распакованная ФС содержит прикладные, сетевые, системные сервисы; файлы настроек, скрипты, служебную информацию и многое другое. Для того чтобы выделить исполняемый код, необходимо провести анализ всех файлов. С помощью утилиты *file* и разработанного многомодульного инструмента на языке

Python проводится сбор информации (расположение, имя файла, аппаратная архитектура, разрядность, размер) обо всех исполняемых файлах, которая заносится в базу данных.

Полученная информация поступает на вход модуля сбора характеристик исполняемого кода. Далее модуль передаёт путь до исполняемого файла на внешний инструмент дизассемблирования IDA Pro, а на выходе получает обработанные данные (Листинг 4).

Листинг 4

```
import os, sys
import binwalk
from subprocess import call, check_output
import subprocess

root_path_firmware = 'dataset/firmware/'
root_path_programm = 'dataset/programm/Go/'

def analisysProgramm ():
    for dirpath, dirnames, filenames in os.walk(root_path_programm):
        for filename in [f for f in filenames]:
            file_out = check_output(['tools/file/file.exe', os.path.join(dirpath, filename)])
            if file_out.find(b'ELF') != -1:
                file_path = file_out.split(b':')[0].decode()
                file_32_64 = file_out.split(b':')[1].split(b',')[0].decode()
                file_platform = file_out.split(b':')[1].split(b',')[1].decode()
                if os.path.isdir(r'report') == False:
                    os.makedirs('report')
                with open(r'report\problemsAnalis.txt', 'a') as fd:
                    fd.write(file_path + ';' + file_32_64 + ';' + file_platform)
                call(['F:/Project/binaryAnalysisProgramm/tools/IDAPro/ida.exe', '-A',
                      '-SF:/Project/binaryAnalysisProgramm/IdaAnalysis.py',      os.path.join(dirpath,
            filename)])

if __name__ == "__main__":
    analisysProgramm ()
```

Математическая основа для расчётов метрик исполняемого кода используется из методов анализа исходных кодов программ. Выделяют следующие типы метрик исходных кодов: количественные метрики, метрики сложности потока управления, метрики сложности потока данных, комбинированные метрики сложности управления и данных, гибридные метрики [102]. На Рисунке 52 представлена классификация метрик исходных кодов программ.

Рисунок 52 — Метрические характеристики программных компонент

В процессе апробации результатов исследования на ЭО применяются адаптированные для исполняемого кода метрики Холстеда [103]. Применение метрических характеристик Холстеда позволяет оценить сложность программных компонент (81), а также количество потенциальных ошибок (83).

Расчёты метрик основаны на следующих исходных данных:

μ_1 — число уникальных операторов;

μ_2 — число уникальных операндов;

N_1 — общее число операторов;

N_2 — общее число операндов.

Стоит отметить, что операторы и операнды в исходных текстах программ содержатся в явном виде. Для того чтобы подсчитать μ_1, μ_2, N_1 и N_2 на основе исполняемых файлов, требуется осуществить дизассемблирование машинного кода. Обладая исходными данными, возможно вычислить словарь

$$\mu = \mu_1 + \mu_2 \tag{79}$$

и длину программы

$$N = N_1 + N_2 \tag{80}$$

Сложность программного кода вычисляется по следующей формуле:

$$S = \left(\frac{\mu_1}{2}\right) * \left(\frac{N_2}{\mu_2}\right) \quad (81)$$

Потенциальное количество ошибок E в коде зависит от объёма программы

$$\mathrm{V} = N * log_2(\mu) \quad (82)$$

$$E = \frac{V}{3000} \quad (83)$$

Сбор характеристик и метрик исполняемого кода реализован в виде единого плагина для IDA Pro.

Собранные характеристики и метрики являются ЭПК для расчётов оценки качества ЭО. В результате обработки 224 файлов прошивки от 112 моделей устройств получено более 185 тыс. записей в БД (Таблица 14).

Таблица 14 — Пример записей в БД

Название программы	Аппаратная платформа	Разрядность	Сложность исполняемого кода	Количество программных функций	Размер	Оценка количества ошибок
6relayd	MIPS	32	222,8	194	42841	85,004
athbox	MIPS	32	272,5	223	182625	193,4
libc.so	Intel 80386	32	561.5	1141	233029	577,1
ppp	Intel 80386	32	454,1	3056	387261	1118,1
dhcp	ARM	32	815,9	1463	176020	360,1
ovpn.ko	ARM	32	369	168	21715	40,2

Полученные данные позволяют экспериментально доказать адекватность предлагаемой аналитической модели, а также провести сравнительный анализ существующих и разработанных методов.

4.2 Анализ результатов применения модели и метода оценки качества КЭ СУ в процессе модернизации сложных систем управления

Доказательство адекватности разработанной аналитической модели оценки качества КЭ СУ основано на проверке корреляции свойств объекта оценивания с оценкой качества. Выделяются два корреляционных отношения:

— увеличение сложности системы снижает её качество;

— увеличение времени на разработку повышает качество системы.

На основе расчётов функциональной пригодности (84) рассмотрим, каким образом меняется зависимость оценки характеристики от исполняемого кода.

$$q_1 = \omega_1^f * \frac{\sum_{i=1}^{|F^c|} \widehat{f_i^c}}{|F^m|} + \omega_2^f * \left(1 - \frac{|F^m \backslash F^c|}{|F^m|}\right) + \omega_3^f * \frac{\sum_{i=1}^{|F^m|} (I_i^c(D) - I_i^m(D))}{|F^m|} + \\ + \omega_4^f * \frac{\sum_{i=1}^{|F^a|} f_i^a}{|F^m|} + \omega_5^f * \frac{\sum_{i=1}^{|F^c|} \widehat{f_i^s}}{|F^c|} \quad (84)$$

Пусть число закодированных функций $|F^c|$ равно числу программных функций в исполняемом коде. Введём следующие ограничения:

— число требуемых функций $|F^m| = 0{,}1 * |F^c|$;

— число нереализованных функций (не закодированных и закодированных неправильно) $|F^m \backslash F^c| = 0{,}1 * |F^m|$;

— число избыточных функций $|F^a| = 0{,}1 * |F^c|$;

— число функций, которые реализованы в соответствии с принятыми стандартами $|F^s| = 0{,}25 * |F^c|$;

— количество ошибок в исполняемом коде распределено поровну между ошибками функциональной пригодности, надёжности и производительности.

Используя статистические данные о распределении ошибок по уровню угроз на систему, рассчитаем количество каждого типа ошибок:

$$E: \begin{cases} E_l = 0{,}62 * E \\ E_m = 0{,}17 * E \\ E_h = 0{,}16 * E \\ E_c = 0{,}05 * E \end{cases}, \quad (85)$$

где E — общее количество ошибок, E_l — количество ошибок с «низкой» угрозой системе, E_m — количество ошибок со «средней» угрозой системе, E_h — количество ошибок с «высокой» угрозой системе, E_c — количество ошибок с «критической» угрозой системе. В соответствии с методом оценки влияния ошибки в ПО на характеристики качества каждому уровню угрозы соответствует числовое значение («низкий» — 0,2; «средний» — 0,5; «высокий» — 0,8; «критический» — 1).

Установленные параметры модели соответствуют наилучшим практикам в разработке для платформы x86_64 (Intel или AMD), где анализ кода интегрирован

на каждом этапе ЖЦ. Результаты расчётов с такими параметрами являются опорными для определения точности предлагаемых модели и методов.

На Рисунке 53 изображён график зависимости оценки качества (по функциональной пригодности) от сложности исполняемого кода (метрика Холстеда).

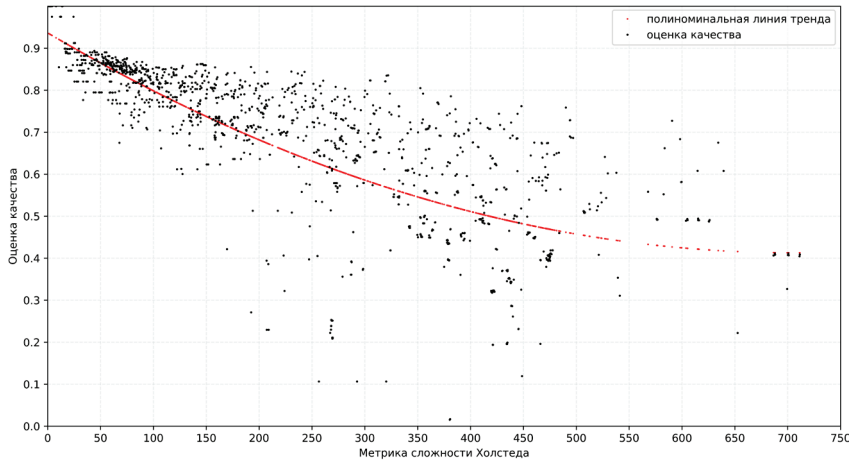


Рисунок 53 — График зависимости оценки качества КЭ СУ от сложности кода

Расчёты функциональной пригодности по данным из ЭО показывают, что усложнение исполняемого кода имеет тенденцию к снижению оценки качества (полиноминальная линия тренда). Следовательно, модель удовлетворяет первому корреляционному отношению.

Проверка второго свойства, которому должна удовлетворять модель, основано на сравнении оценок качества проприетарных программ и свободного ПО. Предположение о низком качестве кода в проприетарном ПО исходит из того, что на его разработку затрачено меньше времени, чем на открытые проекты, которые поддерживаются десятилетиями [104]. В Таблице 15 представлены результаты оценок для одного из производителей КЭ СУ, который в своём ВсПО совместно использует собственное и открытое ПО.

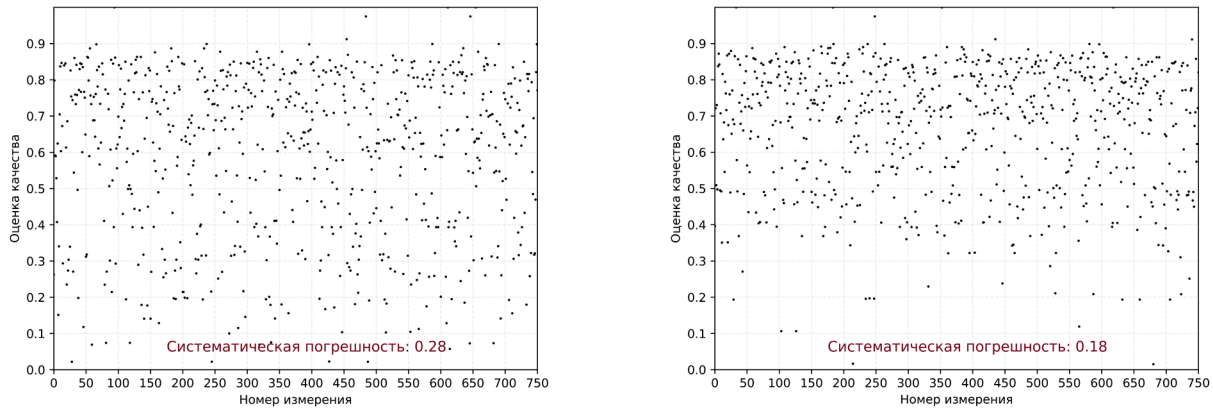
Таблица 15 — Результаты оценок качества

Имя программы	Оценка качества	Тип ПО
ubnt_spectral.ko	0,419	Закрытое ПО (имя программы содержит сокращённое название производителя)
ubnt_poll.ko	0,438	
ubntbox	0,396	
ubnt_poll_host.ko	0,555	
nf_nat_ftp.ko	0,890	ПО под лицензией GPL
iptable_nat.ko	0,867	
i2cdetect	0,850	
tcpdump	0,722	

Анализ результатов, полученных на ЭО, подтверждает состоятельность разработанной аналитической модели, но не доказывает повышение точности оценок качества при переносе ПО на ААП.

В рамках экспериментальной проверки расчёты точности оценки качества КЭ СУ сводятся к критерию «правильность». В случаях оценки качества КЭ СУ при переносе ПО на ААП принять истинное значение не представляется возможным. Но так как значения оценок, полученные в условиях полного цикла разработки систем на базе аппаратной платформы x86_64, основаны на научных принципах и экспериментальных работах, то они приняты в качестве опорных значений. Тогда правильность метода оценки качества КЭ СУ при переносе ПО на ААП определяется систематической погрешностью, которая представлена в главе 1 и выражена формулой (10).

В условиях отсутствия исходных кодов, технической документации или других сведений о КЭ СУ существующие модели оценки способны учитывать только количество возникающих ошибок. Такие модели не являются точными, так как при разработке учитывается влияние ошибок на систему. Предлагаемая модель учитывает степень влияния ошибки в исполняемом коде, а также использует такие ЭПК, которые становятся доступны в ходе анализа исполняемого кода. На Рисунке 54 представлены расчёты оценок качества по функциональной пригодности с использованием существующей (Рисунок 54а) и разработанной (Рисунок 54б) моделей оценки качества.



а) *б)*

Рисунок 54 — Сравнение оценок качества с использованием существующей и разработанной модели

Анализ результатов оценок, полученных при симуляции более 1000 расчётов, доказывает, что применение разработанной модели снижает систематическую погрешность Δ на 10%.

На следующем этапе экспериментальной проверки требуется доказать повышение точности оценки качества КЭ СУ за счёт разработанного метода. Проверка метода оценки качества КЭ СУ включает в себя проверку алгоритма поиска ошибок и проверку методики оценки влияния обнаруженных ошибок на характеристики качества вычислительных систем.

В отличие от существующих, предлагаемый алгоритм поиска ошибок в исполняемом коде разрешает проблемы, которые возникают в случаях тестирования «чёрного» и «серого» ящика. К таким проблемам относятся:

— неполнота тестового покрытия программного кода;

— отсутствие информации о недекларированных возможностях системы;

— отсутствие возможности проводить системное и интеграционное тестирование всех образцов, созданных на разных аппаратных платформах;

— отсутствие возможности выявлять непрогнозируемые отказы;

— неполнота информации о причинах возникновения ошибок в работе КЭ СУ.

Перечисленные проблемы возникают из-за отсутствия возможности применения существующих методов поиска ошибок, так как существующие методы прежде всего основаны на анализе исходного кода программных компонент для платформы x86_64. Моделирование исполняемого кода позволяет описывать программные компоненты (функции, процедуры, отдельные наборы инструкций), связи между ними (графы потока данных и управления) и взаимодействие с аппаратными компонентами (описание работы шин данных, драйверы, системные прерывания) на выбранном формальном языке. Анализ получаемых моделей позволяет выявлять ошибки только по исполняемому коду и при наличии спецификаций на аппаратные компоненты. Применение разработанного алгоритма совместно с существующими методами разрешает проблему отсутствия исходных данных. Возможности алгоритма на практике

представлены в главе 3, тем самым обосновывается теоретическое повышение вероятности обнаружения всех ошибок на 27%.

Второй составной частью метода оценки качества КЭ СУ является методика оценки влияния программной ошибки на характеристики качества. Так как предлагаемая методика представляет собой аналог методики CVSS, то в рамках экспериментальной проверки проведены сравнительные расчёты базовой оценки CVSS и оценки безопасности, рассчитанной на основе нечёткой логики. Оценки совпадают для 75% измерений, что подтверждает адекватность разработанной методики. Основные расхождения обусловлены неоднозначным трактованием промежуточных расчётов. Методика CVSS создана для универсальной оценки уязвимостей без учёта особенностей работы сложных систем управления. В отличие от CVSS, разработанная и представленная методика оценки влияния программной ошибки на характеристики качества учитывает повышенные требования к функциональной пригодности, надёжности и производительности КЭ СУ.

Рассмотрим систему контроля и защиты SCADA-систем от компании ABB. В заявленных функциях системы [105] указано, что применяется стандарт МЭК 61850, включая обмен сообщениями по технологии GOOSE. Указанный стандарт определяет требования к связи для функций и моделей устройств [106]. Отдельно определены требования к доступности устройства по каналам связи и требования к взаимодействию с другими устройствами. Из этого следует, что отсутствие канала связи с устройством контроля и защиты SCADA-систем ведёт к критической ситуации, которая требует незамедлительного решения. Следовательно, ошибка типа «отказ в обслуживании» в программном или аппаратном обеспечении системы имеет наивысший уровень критичности.

Согласно банку данных угроз безопасности информации ФСТЭК России, существует уязвимость BDU:2023-01637 (CVE-2021-22283) [107] типа «отказ в обслуживании» у программно-аппаратных средств контроля и защиты SCADA-систем от компании ABB семейства Relion. Исследователи уязвимости определили базовый вектор по методике CVSS v3: (AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H).

Используя коэффициенты из описания методики CVSS v3 [108] и формулы 1–4, получаем следующие оценки:

$$E_{iss} = 1 - [(1 - 0) * (1 - 0) * (1 - 0{,}56)] = 0{,}56$$

$$E_e = 8.22 * 0{,}55 * 0{,}77 * 0{,}85 * 0{,}85 = 2{,}52$$

$$S = 0$$

$$E_i = 6.42 * 0{,}56 = 3{,}6$$

$$E_{bs} = E_i + E_e = 6{,}1$$

Значение оценки, равное 6,1, соответствует среднему уровню опасности («Medium»), что не требует незамедлительного принятия решения по устранению ошибки в системе. Но стандарт МЭК 61850 предъявляет повышенные требования к устройствам связи, и в данном случае оценка уязвимости должна иметь критический уровень опасности («Critical»).

Таким образом, принятие решений на основе расчётов базовой оценки с помощью аппарата нечёткой логики обосновано, так как для описанной выше ситуации достаточно создать два нечётких правила: в промежуточной оценке «ISS» и окончательной оценке «Base Core». Правила имеют следующий вид:

Листинг 5

```
IF Integrity 'Not' AND Confidentiality 'Not' AND Availability 'High'
TO ISS 'Critical'
IF Impact 'Critical' AND Explotable 'HighCritical' TO CVSS 'Critical'
```

После создания правил уровень критичности ошибки в системе при векторе AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H определяется как критический («Critical»).

Представленная методика имеет гибкую структуру, которая может быть модифицирована под общие или специальные требования. Предлагаемая методика позволяет изменять числовые значения входных термов, что упрощает настройку оценки под конкретную систему. База нечётких правил может быть расширена специальными показателями, которые необходимо учитывать при расчётах.

Вся логика математического аппарата расчёта оценки влияния на безопасность системы запрограммирована на языке Python с использованием библиотеки skfuzzy [109]. В отдельные функции вынесены расчёты оценок FE_{iss}, FE_e, FE_i и FE_{bs}.

Пользователю предоставляется веб-интерфейс для расчёта оценки влияния программной ошибки на систему и для редактирования нечётких правил (Рисунок 55).

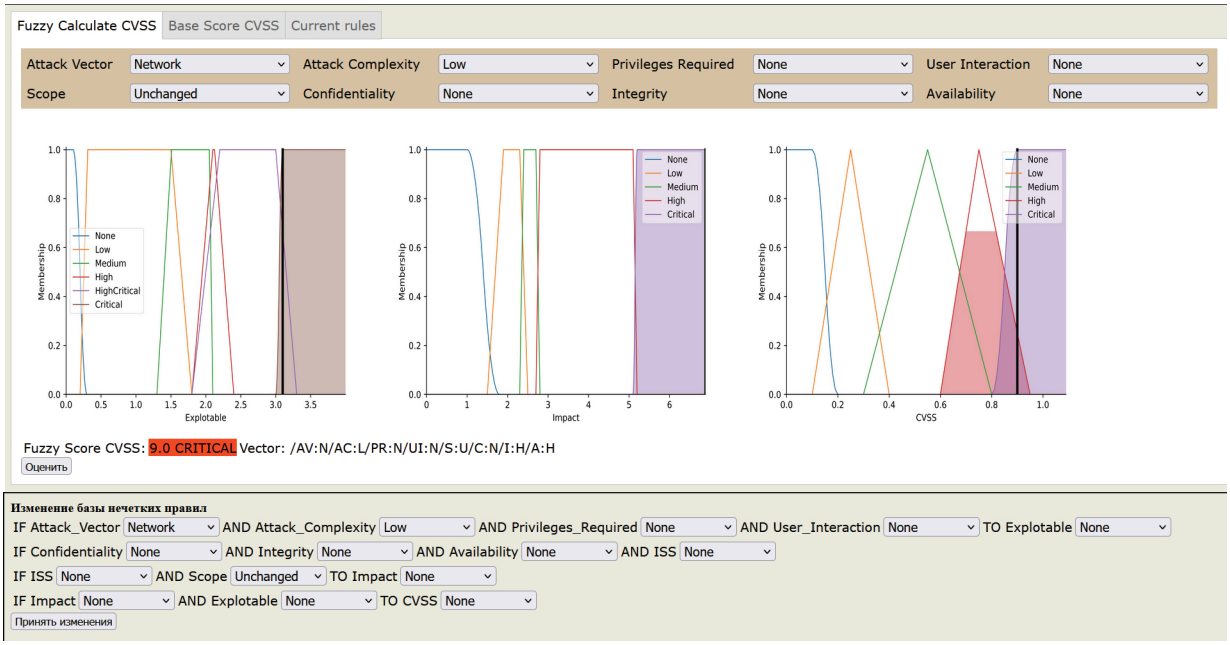


Рисунок 55 — Интерфейс компьютерной программы расчёта оценки влияния ошибки на систему

Разработанная методика обладает гибкостью для внедрения новых метрик и простой базой нечётких правил. К недостаткам можно отнести время расчётов, но в ситуациях, когда необходимо принимать решения о качестве КЭ СУ, времени на расчёт оценки затрачивается намного меньше, чем требуется.

Системное применение разработанного алгоритма, методики и аналитической модели формирует метод оценки качества КЭ СУ. Используя методику расчёта оценки качества, определим систематическую погрешность предлагаемого метода.

На Рисунке 56 представлены результаты оценок качества КЭ СУ по функциональной пригодности с использованием разработанного метода.

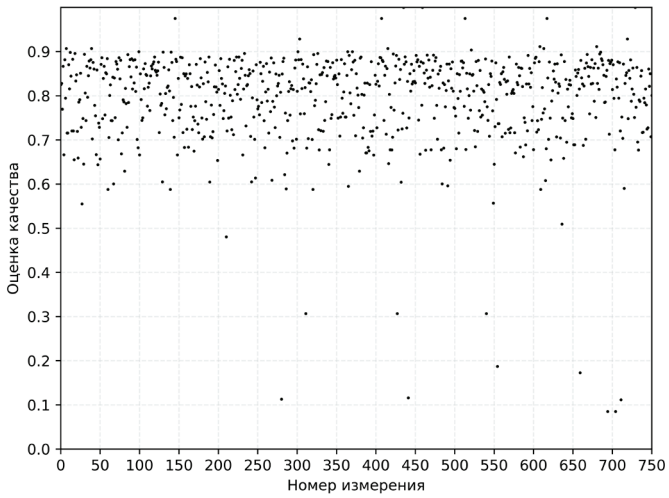


Рисунок 56 — Применение разработанного метода оценки качества КЭ СУ

Из полученных данных математическое ожидание оценок $M(y) = 0{,}830$, принятое опорное значение $\mu = 0{,}792$, тогда систематическая погрешность $\Delta = 0{,}038$.

Диаграмма на Рисунке 57 показывает уровень снижения систематической погрешности за счёт внедрения результатов исследования в процесс переноса ПО на ААП.

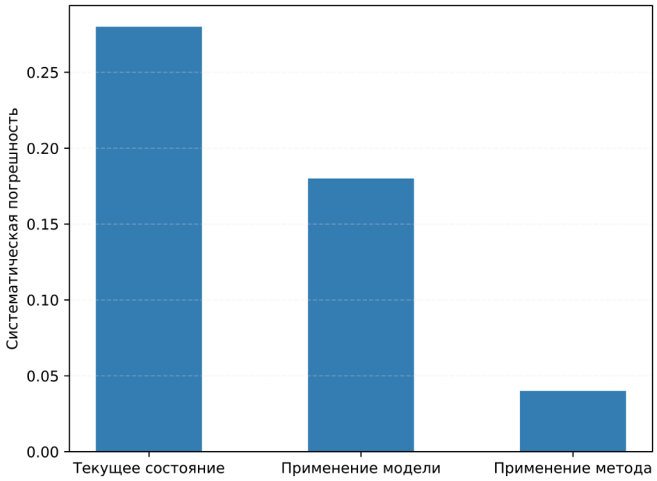


Рисунок 57 — Результаты применения аналитической модели и метода в процессе оценивания качества КЭ СУ

В результате проведённой экспериментальной проверки результатов исследования удалось установить, что применение аналитической модели снижает систематическую погрешность оценки качества на 10%, а применение метода оценки качества КЭ СУ при переносе ПО на ААП — на 24%. Таким образом, удаётся повысить точность в процессе оценивания качества и достичь цели работы.

4.3 Рекомендации по управлению переносом ПО на ААП КЭ СУ с применением предлагаемого метода оценки качества

Анализ видов, последствий и причин отказов в системах управления представляет собой процедуру анализа первоначально предложенной реализации СУ, например с помощью методологии FMEA (от англ. Failure Mode and Effects Analysis), и доработки этой реализации в процессе функционирования системы. Метод FMEA используется на этапах разработки или доработки СУ. Он позволяет предотвратить внедрение в производство недостаточно отработанной системы, помогает улучшить реализацию СУ и заранее предусмотреть необходимые меры в технологии создания, предупреждая появление или снижая комплексный риск системного отказа [110]. Доработка реализации СУ включает в себя модернизацию программно-аппаратных комплексов, в том числе за счёт переноса ПО на ААП. Дерево FMEA образует набор разрешающих правил, по которым лицо, принимающее решение (ЛПР), выбирает одну из множества альтернатив.

При формировании альтернатив для управления переносом ПО на ААП вводятся следующие параметры: оценка качества КЭ СУ (Q), важность КЭ СУ (R) и модифицируемость программного кода КЭ СУ (M).

Многокритериальная оценка качества состоит из оценок функциональной пригодности q_1, надёжности q_2 и производительности q_3. В соответствии с количественным значением оценки определяется качественный уровень каждой из q_n: низкий $[0; 0{,}3)$, средний $[0{,}3; 0{,}7)$ и высокий $[0{,}7; 1]$.

Свёртка многокритериальной оценки в единый качественный показатель Q происходит согласно правилам, представленным в Таблице 16.

Таблица 16 — Определение показателя Q

Q	q_1	q_2	q_3
«неудовлетворительно» (НЕУД)	«низкая»	«низкая»	«низкая»
	«низкая»	«низкая»	«средняя»
	«низкая»	«низкая»	«высокая»
	«низкая»	«средняя»	«низкая»
	«низкая»	«средняя»	«средняя»
	«низкая»	«средняя»	«высокая»

Продолжение таблицы 16

«неудовлетворительно» (НЕУД)	«низкая»	«высокая»	«низкая»
	«низкая»	«высокая»	«средняя»
	«низкая»	«высокая»	«высокая»
	«средняя»	«низкая»	«низкая»
«низкая» (Н)	«средняя»	«низкая»	«средняя»
	«средняя»	«низкая»	«высокая»
	«средняя»	«средняя»	«низкая»
	«средняя»	«средняя»	«средняя»
	«средняя»	«средняя»	«высокая»
	«высокая»	«низкая»	«низкая»
	«высокая»	«низкая»	«средняя»
	«высокая»	«низкая»	«высокая»
«средняя» (С)	«средняя»	«высокая»	«низкая»
	«средняя»	«высокая»	«средняя»
«средняя» (С)	«средняя»	«высокая»	«высокая»
	«высокая»	«средняя»	«низкая»
	«высокая»	«средняя»	«средняя»
	«высокая»	«средняя»	«высокая»
	«высокая»	«высокая»	«низкая»
«высокая» (В)	«высокая»	«высокая»	«средняя»
	«высокая»	«высокая»	«высокая»

Следующим параметром управления переносом является важность R компьютерного элемента в системе управления. Такая оценка рассчитывается на основе конъюнкции или дизъюнкции относительных оценок надёжности всех компонент в зависимости от логического построения системы [111]. Связь элементов СУ может быть последовательной, параллельной, последовательно-параллельной, параллельно-последовательной, через общий элемент («мост»).

В рамках исследования не рассматривается реализация методов расчёта оценки важности. Используя существующие методики, приведём количественные значения к качественным. «Низкая» оценка важности соответствует интервалу $[0; 0{,}2)$, «средняя» - $[0{,}2; 0{,}6)$, «высокая» - $[0{,}6; 0{,}8)$, «критичная» - $[0{,}8; 1]$.

Третьим параметром управления переносом является оценка модифицируемости исполняемого кода программных компонент КЭ СУ. Модернизация программно-аппаратных систем требует учёта затраченных ресурсов на исправление ошибок в программном коде. В тех случаях, когда имеется доступ к исходным кодам и разработка ведётся под одну аппаратную платформу, исправление ошибок осуществляется внесением патчей (программных «заплаток») в программные компоненты. В ситуациях, когда исходные коды отсутствуют или поведение системы на разных платформах отличается, требуется осуществлять

поиск и устранение ошибок в исполняемом коде. Разрешение этой ситуации связано с высокими рисками увеличенных затрат на модернизацию из-за высокой сложности анализа исполняемого кода. Расчёт параметра модифицируемости исполняемого кода позволяет снизить неопределённость при принятии решений о необходимости проведения дополнительных мероприятий.

Модифицируемость зависит от таких показателей исполняемого кода, как понятность и наблюдаемость. Проведённая в работе параметризация кода позволяет использовать в расчётах понятности следующие метрики: целесообразность, согласованность, сложность программного кода (метрики Холстеда, Джилба, Чена, «подсчёт точек пересечения»), открытость кода. Открытость определяется как отношение исполняемого кода, доступного в исходных текстах (например, использование библиотек с открытым исходным кодом), к общему объёму кода.

Метрики сложности исполняемого кода ориентированы на логически связанные, синтаксически и семантически правильные кодовые конструкции без неопределённостей в интерпретации при выполнении. Но анализ кода реальных систем существующими методами позволяет исследовать лишь промежуточные представления, которые являются отображением бинарного кода. Поэтому параметр «наблюдаемость» определяется количеством проблемных мест при отображении бинарного кода в промежуточное представление.

Пусть бинарный код отображается посредством функции дизассемблирования в ассемблерный код (86).

$$\mathrm{d}: B \to A, \tag{86}$$

где B — бинарный код, A — дизассемблированный код, d — функция преобразования (дизассемблирования).

Тогда для того, чтобы отобразить непреобразованный бинарный код $B' \subset B$ в ассемблерный, требуется провести дополнительные мероприятия (87), которые увеличивают затраты на анализ.

$$d': B' \to A \tag{87}$$

Отсюда следует, что наблюдаемость O определяется как отношение количества дизассемблированных данных $V(A)$ (в байтах) к объёму неопределённых данных $V(B')$ (в байтах) (88).

$$O = \frac{V(A)}{V(B')} \tag{88}$$

Чтобы создать градацию уровней параметра наблюдаемости, необходимо провести расчёты зависимости от метрик сложности исполняемого кода. На Рисунке 58 представлены расчёты для аппаратных архитектур Intel, MIPS, PowerPC и ARM.

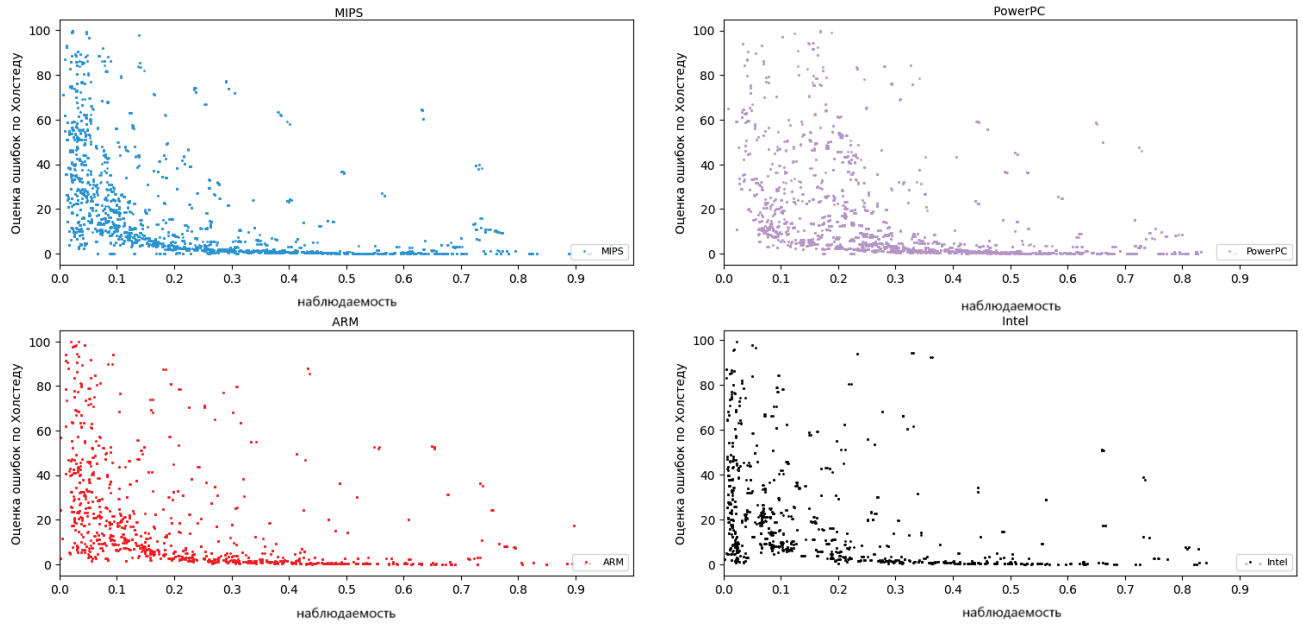


Рисунок 58 — Графики зависимости наблюдаемости от оценок количества ошибок по Холстеду

Графики показывают, что при $O > 0{,}3$ методы автоматического анализа кода испытывают значительные трудности. Следовательно, уровни наблюдаемости имеют следующие интервалы: высокий — [0; 0,1), средний — [0,1; 0,2), низкий — [0,2; 1].

Значения понятности и наблюдаемости для исполняемого кода влияют на количество ошибок, которые возможно исправить. Согласно расчётам установлено, что если модифицируемость кода высокая, то в коде остаётся до 10% ошибок; если средняя, то остаётся до 30 % ошибок; а если низкая, то остаётся более 30% ошибок, что накладывает дополнительные расходы на процесс портирования ПО на ААП.

После определения значений всех параметров управления переносом формируется набор разрешающих правил для ЛПР (Таблица 17).

Таблица 17 — Набор правил для управления процессом модернизации СУ

Оценка			Качество при модернизации и внедрении КЭ СУ	Необходимость в дополнительных мероприятиях
Q	R	М		
В	К	В	Идеальный случай	Нет
В	К	Н	КЭ имеет высокое качество, поэтому отсутствие контроля этого качества не влияет на систему управления	Нет
В	Н	В/Н	КЭ имеет высокое качество и низкую важность в СУ (например, элемент дублирует основной), поэтому модифицируемость не требуется	Нет
С	К	В	При эксплуатации модернизированного КЭ СУ **могут возникать ошибки** в работе СУ, требуется модификация КЭ СУ с целью исправления ошибок в исполняемом коде	Да
С	К	Н	При эксплуатации модернизированного КЭ СУ **могут возникать ошибки** в работе СУ, требуется восстановить исполняемый код и модифицировать КЭ СУ с целью исправления ошибок	Да
С	Н	В	При эксплуатации модернизированного КЭ СУ **могут возникать ошибки** в работе СУ, требуется модифицировать КЭ СУ с целью исправления ошибок	Да
С	Н	Н	При эксплуатации модернизированного КЭ СУ **могут возникать ошибки** в работе СУ, требуется модифицировать КЭ СУ в тех местах, где понятен исполняемый код	Да
Неуд.	К	В	При эксплуатации модернизированного КЭ СУ **будут возникать сбои** в работе СУ, требуется исправить ошибки в исполняемом коде КЭ СУ со значимостью «критичная», «высокая», «средняя» и «низкая»	Да
Неуд.	Н	В	При эксплуатации модернизированного КЭ СУ **могут возникать сбои** в работе СУ, требуется исправить ошибки в исполняемом коде КЭ СУ со значимостью «критичная» и «высокая»	Да
Неуд.	Н	Н	При эксплуатации модернизированного КЭ СУ **могут возникать сбои** в работе СУ, требуется восстановить исполняемый код КЭ СУ со значимостью «критичная» и «высокая» для исправления ошибок	Да
Неуд.	К	Н	Худший вариант. Необходимо остановить внедрение КЭ в СУ. При эксплуатации модернизированного КЭ СУ **будут возникать сбои** в работе СУ. Требуется восстановить исполняемый код КЭ СУ. Исправить ошибки в исполняемом коде КЭ СУ со значимостью «критичная», «высокая», «средняя» и «низкая».	Да

Подводя итог рекомендациям, стоит отметить, что анализ исполняемого кода является трудоёмкой задачей, которая имеет разную степень неопределённости в зависимости от исследуемой системы. Разработка, сопровождение и модернизация программно-аппаратных систем — быстроразвивающаяся отрасль, которая способна перенимать самые передовые достижения науки в кратчайшие сроки. Развитие методов обратного проектирования, в том числе за счёт применения машинного обучения, искусственного интеллекта, во многом способствует поддержанию должного уровня качества для функционирования сложных систем управления.

4.4 Выводы

Проведена и описана экспериментальная проверка результатов исследования. В качестве экспериментального образца использовано телекоммуникационное оборудование как средство приёма, обработки, передачи и хранения информации в системах управления.

Эксперимент подтвердил адекватность предлагаемой аналитической модели, а также установил верность выдвинутых гипотез о том, что усовершенствование аналитической модели качества, применение нового алгоритма поиска ошибок, разработка методики оценки влияния найденных ошибок на систему позволят создать новый высокоточный метод оценки качества компьютерных элементов при модернизации систем управления.

ЗАКЛЮЧЕНИЕ

Цель научно-квалификационной работы достигнута в результате решения поставленных задач:

1) Проведён анализ обобщённых и аналитических моделей качества вычислительных систем, что позволило выявить проблему моделирования компонент системы для расчёта показателей качества в условиях отсутствия исходных кодов. Результаты анализа методов оценки качества свидетельствуют о том, что алгоритмы поиска ошибок в случаях переноса ПО на ААП обнаруживают в среднем на 27% ошибок меньше. Также анализ показывает, что в процессах оценки учитывается только количество ошибок, но не учитывается качество каждой ошибки. Оба этих фактора повышают систематическую погрешность метода оценки качества.
2) Разработана аналитическая модель качества по трём базовым характеристикам (функциональная пригодность, надёжность, производительность). Модель использует 12 ПК, которые в своих расчётах применяют количественные оценки влияния ошибок на характеристики качества.
3) Разработан алгоритм и набор программных инструментов для поиска ошибок и оценки их влияния на характеристики качества. Алгоритм поиска ошибок в исполняемом коде использует существующие методы обратного проектирования, что позволяет обнаружить ошибки в готовом образце системы без информации (технической документации, исходных кодов и др.), требующейся при использовании других методов. Метод оценки влияния ошибки на характеристики качества включает 24 метрики и базу из 363 нечётких правил.
4) Разработан метод оценки качества КЭ СУ, обеспечивающий требуемый уровень точности оценки для эффективного управления процессом модернизации в условиях переноса ПО на ААП, который включает:

- анализ системы с целью выделения значимых функций для оценки качества;
- моделирование программных компонент, которые реализуют выделенные функции системы;
- алгоритм и набор программных инструментов для поиска ошибок в исполняемом коде КЭ СУ;
- метод оценки влияния ошибки на базовые характеристики качества (функциональную пригодность, надёжность, производительность);
- аналитическую модель качества КЭ СУ, которая позволяет рассчитать многокритериальную оценку.

5) Разработанный метод оценки качества КЭ СУ прошёл экспериментальную проверку для телекоммуникационного оборудования, ПО которого переносится на ААП. Подтверждено, что число найденных ошибок в системе возрастает на 27% за счёт возникающей возможности обнаружения аппаратно-зависимых ошибок. Установлено, что при использовании разработанного метода правильность метода оценки качества КЭ СУ при переносе ПО на ААП возрастает на 24%.

Разработанные модель и методы могут быть использованы в процессах управления разработкой новых систем по полному ЖЦ для обеспечения качества и безопасности их использования, что позволит снизить систематическую погрешность методов оценивания на 10%. Также исследования позволяют поддерживать требуемый уровень качества в наследуемых системах, когда техническая информация о них утеряна.

Направление дальнейшего исследования заключается в совершенствовании метода анализа исполняемого кода за счёт применения новых моделей, теории машинного обучения, теории формальной верификации программ. Работа не затрагивает все характеристики качества КЭ СУ, что является точкой роста новых знаний в области системного анализа.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R. Metamodel quality requirements and evaluation (MQuaRE) / arXiv preprint arXiv:2008.09459. – 2020.

2. Балыбердин, В.А. Анализ некоторых подходов к количественной оценке надежности программных средств / А.М. Белевцев А.М., О.А. Степанов //Известия Южного федерального университета. Технические науки. –2015. – №. 11 (172). – С. 157-165.

3. Неборский, С.Н. Оценка надежности обучающих программных средств / С.Н. Неборский //Цифровая трансформация. – 2016. – №. 2. – С. 22-32.

4. Лаврищева, Е.М. Программная инженерия. Парадигмы, технологии и CASE-средства: учебник для вузов / Е.М. Лаврищева. – 2-е изд., испр. и доп. – М.: Издательство Юрайт, 2016. – 280 с.

5. Липаев, В.В. Программная инженерия : методологические основы / В.В. Липаев. – М.|Берлин : Директ-Медиа, 2015. – 608 с.

6. ГОСТ 28195-89. Оценка качества программных средств. Общие положения. М.: ИПК Издательство стандартов, 1989. 31 с.

7. ГОСТ Р ИСО/МЭК 25010-2015. Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения. - М.: Стандартинформ, 2015. - 36 с.

8. ГОСТ Р. ИСО/МЭК 25021-2014 Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Элементы показателя качества. - М.: Стандартинформ, 2014. – 51 с.

9. ГОСТ Р ИСО/МЭК 25045-2015 Информационные технологии (ИТ). Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модуль оценки восстанавливаемости. М.: Стандартинформ, 2016. – 41 с.

10. Нарышкин, К.В. Факторы возникновения неконтролируемого поведения информационных систем при трансфере технологий / К. В. Нарышкин //

Сборник докладов научно-практической конференции «Цифровые технологии и их приложения», ЦТиМ ФГБОУ ВО «УГНТУ», Уфа, 2021. [Электронная версия].

11. Зацаринный, А.А. О применении экспертных методов при оценке эффективности и качества информационных систем / А.А. Зацаринный, Ю.С. Ионенков // Системы и средства информатики. – 2022. – Т. 32, № 2. – С. 47-57. – DOI 10.14357/08696527220205.

12. Таран, В.Н. Анализ требований при проектировании информационной системы / В.Н. Таран, О.Ю. Савченко, И.А. Максимова-Федорцова // Информационные системы и технологии в моделировании и управлении : Сборник материалов III Всероссийской научно-практической конференции с международным участием, посвященной 100-летию Крымского федерального университета имени В.И. Вернадского , Ялта, 21–23 мая 2018 года / Ответственный редактор К.А. Маковейчук. – Ялта: Общество с ограниченной ответственностью «Издательство Типография «Ариал», 2018. – С. 398-403.

13. Щенников, А.Н. Качество информационных систем / А.Н. Щенников // ИТНОУ: Информационные технологии в науке, образовании и управлении. – 2018. – № 1(5). – С. 53-62.

14. Basili, V. et al. Aligning Organizations Through Measurement: The GQM+ Strategies Approach. – Berlin : Springer International Publishing, 2014. – С. 205.

15. Методический подход к построению моделей прогнозирования показателей свойств систем информационной безопасности / П.Д. Зегжда, А.Ф. Супрун, В.Г. Анисимов [и др.] // Проблемы информационной безопасности. Компьютерные системы. – 2019. – № 4. – С. 45-49.

16. Большаков, А.С. Программное обеспечение моделирования угроз безопасности информации в информационных системах / А.С. Большаков, Д.И. Раковский // Правовая информатика. – 2020. – № 1. – С. 26-39. – DOI 10.21681/1994-1404-2020-1-26-39.

17. Нарышкин, К.В. Модель качества иностранной информационной системы / К.В. Нарышкин // Российская наука в современном мире : Сборник статей XLVIII международной научно-практической конференции, Москва, 31 августа

2022 года. – Москва: Общество с ограниченной ответственностью "Актуальность.РФ", 2022. – С. 25-28.

18. Баррет С., Пак Д. Встраиваемые системы. Проектирование приложений на микроконтроллерах семейства 68HC12/HCS12 с применением языка C. – Litres, 2022.

19. Вертемягин, А.В. Оценка производительности объектов вычислительных систем и сетей при различных уровнях их живучести / А.В. Вертемягин, В.Г. Литвин, Ю.В. Литвин // Автоматизация процессов управления. – 2021. – № 2(64). – С. 77-85. – DOI 10.35752/1991-2927-2021-2-64-77-85.

20. Ефимов, С.Н. Алгоритм нахождения вероятности функционирования с заданной производительностью аппаратно-программного комплекса систем реального времени / С.Н. Ефимов, В.А. Терсков, О.Ю. Серикова // Вестник Воронежского института МВД России. – 2021. – № 1. – С. 72-81.

21. Каштанов, В.А. Теория надежности сложных систем / В.А. Каштанов, А.И. Медведев. – Москва : Физматлит, 2010. – 608 с. – ISBN 978-5-9221-1132-4.

22. Thomas, J. McCabe, “A Complexity Measur”, IEEE Transactions on Software Engineering SE-2, 308–320 (1976).

23. Myers, G., “An Extension to the Cyclomatic Measure of Programm Complexity”, SIGPLAN Notices, October 1977

24. Холстед, М.Х. Начала науки о программах / Пер. с англ. М.: Финансы и статистика, 1981. 128 с.

25. Гончаров, А.Н. Об одном из подходов к измерению качества программного кода имитационных моделей комплексной испытательной моделирующей установки / А.Н. Гончаров, В.С. Колесников, А.П. Демченко // Проблемы повышения эффективности научной работы в оборонно-промышленном комплексе России : Материалы V Всероссийской научно-практической конференции, Знаменск, 24–25 марта 2022 года / Сост.: С.Н. Бориско. – Астрахань: Федеральное государственное бюджетное образовательное учреждение высшего образования «Астраханский государственный университет», 2022. – С. 18-26.

26. Звездин, С.В. Проблемы измерения качества программного кода / С.В. Звездин // Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника. – 2010. – № 2(178). – С. 62-66.

27. Официальный сайт компании Scientific Toolworks, Inc. Understand Features. - URL: https://scitools.com/features (дата обращения: 13.04.2023).

28. Диасамидзе, С.В. Метод выявления недекларированных возможностей программ с использованием структурированных метрик сложности: дис. … канд. техн. наук: 05.13.19 / Диасамидзе Светлана Владимировна. – СПб, 2012. – 161 с.

29. Викторов, Д.С. Методика статического анализа для поиска дефектов естественной семантики программных объектов и ее программная реализация на базе инфраструктуры компилятора LLVM и фронтенда Clang / Д.С. Викторов, Е.Н. Жидков, Р.Е. Жидков // Журнал СФУ. Техника и технологии. – 2018. – № 11(7). – С. 801-810.

30. Graham B., Leroux P. N., Landry T. Using Static and Runtime Analysis to Improve Developer Productivity and Product Quality //white paper, QNX Software Systems. – 2008.

31. Каушан, В.В. Поиск ошибок выхода за границы буфера в бинарном коде программ: дисс. … канд. техн. наук: 05.13.11 / Каушан Вадим Владимирович. – М, 2018. – 92 с.

32. Ермаков, М.К. Проведение динамического анализа исполняемого кода формата ARM ELF на основе статического бинарного инструментирования / М.К. Ермаков // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. – 2016. – № 1(236). – С. 108-117. – DOI 10.5862/JCSTCS.236.10.

33. Хамадулин, Э.Ф. Методы и средства измерений в телекоммуникационных системах : Учебное пособие / Э.Ф. Хамадулин. – 1-е изд.. – Москва : Издательство Юрайт, 2018. – 365 с. – (Бакалавр. Академический курс). – ISBN 978-5-9916-5976-5.

34. Соя, Д. Входной контроль печатных плат. Виды дефектов / Д. Соя, М. Степанищев // Электроника: Наука, технология, бизнес. – 2022. – № 3(214). – С. 166-172. – DOI 10.22184/1992-4178.2022.214.3.166.170.

35. Дианов, В.Н. Перспективы использования нейронных сетей для диагностики сбоев электронной аппаратуры / В.Н. Дианов // Известия Московского государственного индустриального университета. – 2007. – № 2(7). – С. 46-50.

36. Микони, С.В. Общие диагностические базы знаний вычислительных систем //СПб.: СПИИРАН. – 1992. – С. 234.

37. Криспин, Лайза, Грегори, Джанет. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2010. — 464 с. : ил. — Парал. тит. англ. ISBN 978-5-8459-1625-9 (рус.)

38. Дрожжин, А.В. чем суть скандала с дизельными двигателями Volkswagen. Блог компании АО «Лаборатория Касперского». - URL: https://www.kaspersky.ru/blog/dieselgate-explained/10406/ (дата обращения: 01.05.2022)

39. Куликов, С.С. Тестирование программного обеспечения : учеб. пособие / С.С. Куликов , Г.В. Данилова, О.Г. Смолякова, М.М. Меженная. – Минск : БГУИР, 2019. – 276 с. : ил.

40. Значения метрик кода. Техническая документация Майкрософт - URL: https://docs.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022 (дата обращения: 01.05.2022)

41. Колташев, А.А. Основные принципы системного тестирования и подтверждения бортового программного обеспечения спутников / А.А. Колташев // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева. – 2010. – № 1(27). – С. 4-7.

42. Чупилко, М.М. Автоматизация системного тестирования моделей аппаратуры на основе формальных спецификаций / М.М. Чупилко // Труды Института системного программирования РАН. – 2010. – Т. 18. – С. 115-128.

43. Кустов, Д.А. Анализ тестирования программного обеспечения методом белого ящика и методом черного ящика / Д.А. Кустов // Научный аспект. – 2024. – Т. 12, № 5. – С. 1541-1546.

44. NIST Special Publication 800-142. Practical Combinatorial Testing. - URL: http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf (дата обращения: 18.04.2023)

45. Владимиров, М.А. Критерии полноты тестового покрытия в генетических алгоритмах генерации тестов / М.А. Владимиров // Труды Института системного программирования РАН. – 2006. – Т. 9. – С. 57-66.

46. Золотухина, Е.Б. Обзор методов тестирования программного обеспечения / Е.Б. Золотухина, Е.А. Макарова, А.А. Беляков // Аллея науки. – 2018. – Т. 4, № 6(22). – С. 10-18.

47. Берман, А.Ф. Метод синтеза и анализа деревьев отказов на основе понятий механизма и кинетики событий / А.Ф. Берман, Н.Ю. Павлов, О.А. Николайчук // Проблемы анализа риска. – 2018. – Т. 15, № 3. – С. 62-77.

48. Банк данных угроз безопасности: сайт ФСТЭК России. - URL: https://bdu.fstec.ru/threat (дата обращения: 12.06.2022).

49. Карбовский, С.В. Несовершенство понятийного базиса технологий обратного проектирования в жизненном цикле сложных систем / С.В. Карбовский // International Journal of Open Information Technologies. – 2023. – Т. 11, № 6. – С. 38-45.

50. Свидетельство о государственной регистрации программы для ЭВМ № 2015614974 Российская Федерация. Программа для формирования и применения сигнатур вызовов функций в среде дизассемблера IDA Pro : № 2015610122 : заявл. 12.01.2015 : опубл. 05.05.2015 / В.А. Новиков.

51. Новиков, В.А. Рекомпиляция дизассемблированных текстов программ / В.А. Новиков, М.О. Фонарев // Вопросы защиты информации. – 2007. – № 2(77). – С. 51-54.

52. Буйневич, М.В. Исследование возможности применения машинного обучения для поиска уязвимостей в программном коде в процессе его статического анализа / М.В. Буйневич, К.Е. Израилов // Интеграция науки, общества,

производства и промышленности: проблемы и перспективы : сборник статей Всероссийской научно-практической конференции, Челябинск, 17 апреля 2020 года. – Уфа: Общество с ограниченной ответственностью "Аэтерна", 2020. – С. 17-22.

53. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем //Препринт ИСП РАН. – 2016. – Т. 29. – С. 1-52.

54. Статический анализатор Svace для поиска дефектов в исходном коде программ / В.П. Иванников, А.А. Белеванцев, А.Е. Бородин [и др.] // Труды Института системного программирования РАН. – 2014. – Т. 26, № 1. – С. 231-250.

55. Белеванцев, А.А. Анализ сущностей программ на языках Си/Си++ и связей между ними для понимания программ / А.А. Белеванцев, Е.А. Велесевич // Труды Института системного программирования РАН. – 2015. – Т. 27, № 2. – С. 53-64. – DOI 10.15514/ISPRAS-2015-27(2)-4.

56. Тарасов, Д.А. Проблемы установления авторства программного продукта в рамках компьютерно-технической экспертизы / Д.А. Тарасов // Теория и практика судебной экспертизы: международный опыт, проблемы, перспективы : сборник научных трудов I Международного форума, Москва, 07–08 июня 2017 года. – Москва: Московский университет Министерства внутренних дел Российской Федерации им. В.Я. Кикотя, 2017. – С. 601-606.

57. Bouchiha, D. Reengineering Legacy Systems Towards New Technologies //Encyclopedia of Information Science and Technology, Fifth Edition. – IGI Global. – С. 1214-1230.

58. Tamburri, D.A., Kazman R. General methods for software architecture recovery: a potential approach and its evaluation //Empirical Software Engineering. – 2018. – Т. 23. – №. 3. – С. 1457-1489.

59. Zhang, G., Zhou J. The effects of forward and reverse engineering on firm innovation performance in the stages of technology catch-up: An empirical study of China //Technological Forecasting and Social Change. – 2016. – Т. 104. – С. 212-222.

60. Dehaghi, M.R., Goodarzi M. Reverse engineering: a way of technology transfer in developing countries like Iran //International Journal of e-Education, e-Business, e-Management and e-Learning. – 2011. – Т. 1. – №. 5. – С. 347.

61. Нарышкин, К.В. Описательная модель системы обратного проектирования программного обеспечения / К. В. Нарышкин // Приоритетные направления инновационной деятельности в промышленности : сборник научных статей V международной научной конференции в 2-х частях, Казань, 30–31 мая 2021 года / НПП МЕДПРОМДЕТАЛЬ ООО Газпром трансгаз Казань. Том Часть 2. – Казань: Общество с ограниченной ответственностью "КОНВЕРТ", 2021. – С. 51-53.

62. Нарышкин, К.В. Обратное проектирование как метод системного анализа бинарного кода / К.В. Нарышкин // Сборник тезисов докладов VIII научно-технической конференции «Математическое моделирование, инженерные расчеты, и программное обеспечение для решения задач ВКО», Москва, 2023. - с.32-33.

63. Wang, X., Zeldovich N., Kaashoek M.F., Solar-Lezama A.A Differential Approach to Undefined Behavior Detection. ACM Transactions on Computer Systems, vol. 33, no. 1, art. 1, 2015, 29 p. DOI: 10.1145/2699678.

64. Lattner, C., Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In Proc. of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, 2004, pp. 75-86.

65. Соловьев, М.А. О новом поколении промежуточных представлений, применяемых для анализа бинарного кода / М.А. Соловьев, М. Г. Бакулин, М. С. Горбачев [и др.] // Труды Института системного программирования РАН. – 2018. – Т. 30, № 6. – С. 39-68. – DOI 10.15514/ISPRAS-2018-30(6)-3.

66. Song, D., Brumley D., Yin H., Caballero J., Jager I., Kang M.G., Liang Z., Newsome J., Poosankam P., Saxena P. BitBlaze: A new approach to computer security via binary analysis. Information systems security, 2008, pp. 1-25.

67. Brumley, D., Jager I., Avgerinos T., Schwartz E.J. BAP: a binary analysis platform. Computer Aided Verification, 2011, pp. 463-469.

68. Dullien, T., Porst S. REIL: A platform-independent intermediate representation of disassembled code for static code analysis. In Proc. of the CanSecWest Conference, 2009.

69. Авербух, А.И. применение стандартных образцов для оценивания точности (правильности и прецизионности) методик выполнения измерений / А.И. Авербух, О.В. Кочергина // Стандартные образцы. – 2008. – № 3. – С. 33-37.

70. Герасимов, А.Ю. Классификация предупреждений о программных ошибках методом динамического символьного исполнения программ дисс. … канд. физ.-мат. наук: 05.13.11 / Герасимов Александр Юрьевич, 2019. – 129 с.

71. Нарышкин, К.В. Кластеризация отчетов об исправлениях в программном обеспечении сетевого оборудования / К.В. Нарышкин // Вестник Концерна ВКО "Алмаз – Антей". – 2023. – № 2. – С. 90-100. – DOI 10.38013/2542-0542-2023-2-90-100.

72. Нарышкин, К.В. Кластерный анализ открытых отчетов об ошибках программного обеспечения встраиваемых систем / К.В. Нарышкин // Сборник тезисов докладов VII научно-технической конференции «Математическое моделирование, инженерные расчеты, и программное обеспечение для решения задач ВКО», Москва, 2022. - с.44.

73. Нарышкин, К.В. Анализ отчетов изменений в программном обеспечении встраиваемых систем /В.А. Новиков, К.В. Нарышкин // Актуальные тренды цифровой трансформации промышленных предприятий : сборник статей Всероссийской научно-практической конференции, Казань, 21–24 сентября 2022 года. – Курск: Закрытое акционерное общество "Университетская книга", 2022. – С. 204-208.

74. Нарышкин, К.В. Анализ открытой информации об исправлениях в программном обеспечении встраиваемых систем / К.В. Нарышкин // Актуальные проблемы прикладной информатики в образовании, экономике, государственном и муниципальном управлении: Материалы Международной научной конференции, Барнаул, 25 мая 2022 года / Под редакцией А.Ю. Юдинцева, Г.Н. Трошкиной. Том Выпуск VII. – Барнаул: Алтайский государственный университет, 2022. – С. 72-76.

75. SEI CERT «Стандарты программирования». – URL: https://wiki.sei.cmu.edu/confluence/ (дата обращения: 19.09.2023).

76. Кондратьев, В.В. Модельно-ориентированный системный инжиниринг 2.0 : учеб. пособие / В.В. Кондратьев. – Москва : МФТИ, 2021. – 102 с. : ил. ; Библиогр. 11 назв. ISBN 978-5-7417-0779-1

77. Нарышкин, К.В. Анализ моделей оценки качества вычислительной системы / К.В. Нарышкин // International Journal of Open Information Technologies. – 2023. – Т. 11, № 10. – С. 44-54.

78. Хубаев, Г.Н. Сравнение сложных программных систем по критерию функциональной полноты / Г.Н. Хубаев // Программные продукты и системы. – 1998. – № 2. – С. 6-9.

79. Новиков, В.А. "Спецпроверка" программ / В.А. Новиков, Р.И. Компаниец, А.Г. Ломако // Защита информации. Инсайд. – 2006. – № 3(9). – С. 18-27.

80. Калайда, В.Т. Технология разработки программного обеспечения : учебное пособие / В.Т. Калайда, В.В. Романенко ; В.Т. Калайда, В.В. Романенко ; Федеральное агентство по образованию, Томский гос. ун-т систем упр. и радиоэлектроники. – Томск : ТУСУР, 2007. – 237 с. – ISBN 978-5-86889-336-0.

81. Стрелавина, О.Д. Повышение надежности программного обеспечения для распределенных систем управления / О.Д. Стрелавина, С.Н. Ефимов, В.А. Терсков, М.А. Лихарев // Сибирский аэрокосмический журнал. – 2021. – Т. 22, № 3. – С. 459-467. – DOI 10.31772/2712-8970-2021-22-3-459-467.

82. Агамирзян, И.Р. Открытые стандарты и совместимость программных систем //Информационное общество. – 2005. – №. 2. – С. 55-56.

83. Черкесов, Г.Н. Надежность аппаратно-программных комплексов: учеб. пособие для студентов вузов, обучающихся по направлению подгот. дипломир. специалистов 654600 "Информатика и вычислит. техника" и направлению подгот. бакалавров , 552800 "Информатика и вычислит. техника" / Г.Н. Черкесов ; Г.Н. Черкесов. – СПб. [и др.]: Питер, 2005. – 478 с. – (Учебное пособие). – ISBN 5-469-00102-4.

84. Панков, Д.А. Способы и алгоритмы тестирования программно-аппаратных комплексов на основе имитации неисправностей: дисс. канд. техн. наук: 05.13.01 / Панков Денис Анатольевич, 2021. – 153 с.

85. Шкляр, В.Н. Надежность систем управления: учебное пособие / В.Н. Шкляр; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2009. – 126 с.

86. Вентцель Е.С. Теория вероятностей: Учеб. для вузов. — 6-е изд. стер. — М.: Высш. шк., 1999.— 576 с.

87. Лоскутов, А.В. Математическое моделирование вычислительной системы на заказной СБИС на основе аппарата систем массового обслуживания с отказами / А.В. Лоскутов, А.Н. Кривоногов, А.В. Погосян, А.В. Боронин // REDS: Телекоммуникационные устройства и системы. – 2017. – Т. 7, № 3. – С. 326-330.

88. Amurrio, A. Response-Time Analysis of Multipath Flows in Hierarchically-Scheduled Time-Partitioned Distributed Real-Time Systems / A. Amurrio, E. Azketa, J. Gutierrez, M. Aldea, M. G. Harbour // IEEE Access. 2020. T. 8. P. 196700–196711.

89. Omran, S.S., Jumma L.F. Design SHA-2 MIPS Processor Using FPGA //Conference of Cihan University-Erbil on Communication Engineering and Computer Science. – 2017. – С. 14.

90. Briand, L.C., Labiche Y., Leduc J. Toward the reverse engineering of UML sequence diagrams for distributed Java software //IEEE Transactions on Software Engineering. – 2006. – Т. 32. – №. 9. – С. 642-663.

91. Lehtinen, N. Error functions Электронный ресурс. 2010. URL: http://nlpc.stanford.edu/nleht/Science/reference/errorfun.pdf (дата обращения: 20.10.2023).

92. Нарышкин, К.В. Модельно-ориентированное проектирование в системе оценки качества телекоммуникационного оборудования / К.В. Нарышкин // International Journal of Open Information Technologies. – 2024. – Т. 12, № 9. – С. 30-39.

93. Зайцев, Д.А. Парадигма вычислений на сетях Петри / Д.А. Зайцев // Автоматика и телемеханика. – 2014. – № 8. – С. 19-36.

94. Нарышкин, К.В. Метод анализа пространства состояний программы при поиске уязвимостей в исполняемом коде / К.В. Нарышкин // Кибербезопасность: технические и правовые аспекты защиты информации: Сборник научных трудов III Национальной научно-практической конференции,

Москва, 23–24 апреля 2024 года. – Москва: МИРЭА - Российский технологический университет, 2024. – С. 18-23.

95. Общая система оценки уязвимостей версии 3.1: Спецификация стандарта: сайт FIRST. - URL: https://www.first.org/cvss/v3.1/specification-document (дата обращения: 14.09.2022).

96. Horváth, A., Erdősi P.M., Kiss F. The common vulnerability scoring system (cvss) generations–usefulness and deficiencies. – 2016.

97. История создания CVSS-SIG версия 2: Спецификация стандарта: сайт FIRST. - URL: https://www.first.org/cvss/v2/history (дата обращения: 17.02.2023).

98. Нарышкин, К.В. Методика оценки влияния программной ошибки на систему с помощью нечеткой логики / К.В. Нарышкин // Автоматизация в промышленности. – 2023. – № 8. – С. 59-64. – DOI 10.25728/avtprom.2023.08.12.

99. Нарышкин, К.В. Алгоритм подготовки к инструментации встроенного программного обеспечения / С.В. Карбовский, К.В. Нарышкин // Кибербезопасность: технические и правовые аспекты защиты информации : Сборник научных трудов I Национальной научно-практической конференции, Москва, 24–26 мая 2023 года. – Москва: МИРЭА - Российский технологический университет, 2023. – С. 34-37.

100. Нарышкин, К.В. Инструменты сравнения бинарного кода / К. В. Нарышкин // Системный администратор. – 2023. – № 12(253). – С. 90-94.

101. Карбовский, С.В. Методика экспертизы программного обеспечения на основе усовершенствованного алгоритма расчета скользящей энтропии двоичных файлов / С.В. Карбовский // International Journal of Open Information Technologies. – 2023. – Т. 11, № 7. – С. 53-59.

102. Ледовских, И.А. Метрики сложности кода / И.А. Ледовских ; Технический отчет 2012-2. – 2012. – 11 с.

103. Аверьянов, А.В. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ / А.В. Аверьянов, И.Н. Кошель, В.В. Кузнецов // Известия высших учебных заведений. Приборостроение. – 2019. – Т. 62, № 11. – С. 970-975. – DOI 10.17586/0021-3454-2019-62-11-970-975.

104. Alexopoulos N. et al. How long do vulnerabilities live in the code? a {Large-Scale} empirical measurement study on {FOSS} vulnerability lifetimes //31st USENIX Security Symposium (USENIX Security 22). – 2022. – С. 359-376.

105. Устройство управления и защиты фидеров REF611: официальный сайт компании ABB. - URL: https://new.abb.com/medium-voltage/ru/reshenia-dlya-avtomatizatsii-raspred-seti/tsifroviye-rele/upravleniye-i-zaschita-fidera/ustroistvo-ref611 (дата обращения 28.07.2023).

106. ГОСТ Р ИСО/МЭК 61850-5-2011 Сети и системы связи на подстанциях. Часть 5. Требования к связи для функций и моделей устройств. – М.: Стандартинформ, 2020 – 130 с.

107. Банк данных угроз безопасности: сайт ФСТЭК России. - URL: https://bdu.fstec.ru/vul/2023-01637 (дата обращения: 01.08.2023).

108. Общая система оценки уязвимостей версии 3.1: Спецификация стандарта: сайт FIRST. - URL: https://www.first.org/cvss/v3.1/specification-document (дата обращения: 14.09.2022).

109. Нарышкин, К.В. Свидетельство о государственной регистрации программы для ЭВМ № 2023669264 Российская Федерация. Программа расчета оценки влияния программной ошибки на вычислительную систему: № 2023668181: заявл. 28.08.2023: опубл. 12.09.2023 / К.В. Нарышкин.

110. Мартынюк, А.В. FMEA-анализ как один из комплексных методов эффективного управления качеством / А.В. Мартынюк, А.В. Зарецкий, Т.И. Зимина, М.А. Макаров // Актуальные проблемы гуманитарных и естественных наук. – 2012. – № 6. – С. 122-126.

111. Gao R., Yao K. Importance index of components in uncertain reliability systems // Journal of Uncertainty Analysis and Applications. – 2016. – Т. 4. – С. 1-19.

Приложение А

Акты реализации диссертационных исследований

МИНОБРНАУКИ РОССИИ
Федеральное государственное
бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
просп. Вернадского, д. 78, Москва, 119454
тел.: (499) 215 65 65 доб. 1140, факс: (495) 434 92 87

УТВЕРЖДАЮ
Заместитель первого проректора
_____ Ю.А. Ефимова
«16» октября 2024 г.

АКТ
о внедрении (использовании) результатов диссертационной работы Нарышкина Константина Викторовича на тему «Метод оценки качества компьютерных элементов системы управления при переносе программного обеспечения на альтернативные аппаратные платформы»

Материалы диссертационной работы Нарышкина Константина Викторовича на тему «Метод оценки качества компьютерных элементов системы управления при переносе программного обеспечения на альтернативные аппаратные платформы», представленной на соискание ученой степени кандидата технических наук по специальности 2.3.1.Системный анализ, управление и обработка информации, статистика, используются в учебном процессе Института кибербезопасности и цифровых технологий на кафедре «Интеллектуальные системы информационной безопасности» РТУ МИРЭА для подготовки специалистов по специальности 10.05.05 «Безопасность информационных технологий в правоохранительной сфере».

Председатель комиссии:
Директор
Института кибербезопасности и
цифровых технологий А.А. Бакаев

Члены комиссии:
Заместитель директора
Института кибербезопасности и
цифровых технологий О.А.Глобенко

Заведующий кафедрой
«Интеллектуальные системы информационной безопасности» Е.А. Максимова

АО «АСТ»
115230, г. Москва, Каширское шоссе, д. 3, корпус 2, стр. 4
123242, г. Москва, пер. Капранова, д. 3, стр. 2
+7 (495) 679-86-86
hello@acti.ru
Advanced System Technologies
www.acti.ru

Исх. № 451-619/24

УТВЕРЖДАЮ:
Генеральный директор
АО «АСТ»
И.А. Зотов
«02» ноября 2024 г.

АКТ
о реализации научных результатов, полученных Нарышкиным Константином Викторовичем

г. Москва «02» ноября 2024 г.

Комиссия в составе:

- Левыкина Михаила Владимировича, кандидата технических наук, руководителя управления исследований и разработки;
- Авраменко Николая Евгеньевича, руководителя проектов;
- Скиданова Михаила Андреевича, главного инженера направления специальных исследований

подтверждает, что научные результаты, полученные Нарышкиным Константином Викторовичем, а именно:

1) аналитическая модель качества компьютерных элементов сложной системы управления;
2) методика и программа расчета оценки влияния программной ошибки на вычислительную систему;
3) метод и алгоритм оценки качества компьютерных элементов сложной системы управления;

использованы при подготовке общей научно-технической документации по 1 этапу НИР, выполняемой на основании ГК №24/173 от 26/04/2024.

Руководитель управления, канд. техн. наук _____ М.В. Левыкин

Руководитель проектов _____ Н.Е. Авраменко

Главный инженер направления _____ М.А. Скиданов

Приложение Б

Свидетельство о государственной регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2023669264

Программа расчета оценки влияния программной ошибки на вычислительную систему

Правообладатель: ***Нарышкин Константин Викторович (RU)***

Автор(ы): ***Нарышкин Константин Викторович (RU)***

Заявка № **2023668181**

Дата поступления **28 августа 2023 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***12 сентября 2023 г.***

Руководитель Федеральной службы по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 429b6a0fe3853164baf96f83b73b4aa7
Владелец Зубов Юрий Сергеевич
Действителен с 10.05.2023 по 02.08.2024

Ю.С. Зубов