DOI: 10.55648/1998-6920-2023-17-1-92-106 УДК 004.05

Обзор методов верификации и оценки качества программного обеспечения

Е. Ю. Мерзлякова, Е. В. Янченко

Сибирский гос. унив. телекоммуникаций и информатики (СибГУТИ)

Аннотация: Определены основные понятия в области тестирования и оценки качества программного обеспечения (ПО). В частности, рассмотрены понятия верификации, валидации и жизненного цикла программного обеспечения и их роль в процессе тестирования. Рассмотрено понятие артефакта в жизненном цикле ПО. Взяты во внимание российские и международные стандарты в области оценки качества ПО. Рассмотрены существующие методы верификации ПО: экспертиза, статический анализ, формальные методы, динамические и синтетические методы. Сделан обзор методов оценки качества ПО, используемых в циклах разработки и тестирования программ. Также рассмотрены функциональные и административные факторы, влияющие на качество ПО. В статье перечислены современные средства автоматизации тестирования, которые могут использоваться на сегодняшний день. Обзорный материал статьи будет полезен для разработчиков ПО, начинающих тестировщиков и смежных специалистов.

Ключевые слова: тестирование ПО, верификация, валидация, анализ программ, жизненный цикл ПО, автоматизация тестирования, оценка качества, методы верификации ПО, разработка программ, качество программных средств.

Для цитирования: Мерзлякова Е. Ю., Янченко Е. В. Обзор методов верификации и оценки качества программного обеспечения // Вестник СибГУТИ. 2023. Т. 17, № 1. С. 92–106. https://doi.org/10.55648/1998-6920-2023-17-1-92-106.

Статья поступила в редакцию 17.10.2022;
переработанный вариант – 30.01.2023;
принята к публикации 10.02.2023.

1. Введение

В настоящее время информационные технологии прочно переплетаются со всеми сферами жизни общества. Каждый день многие люди работают с различным программным обеспечением, и его сложность достаточно широко варьируется от небольших прикладных программ до критического программного обеспечения. Сегодня мы используем различные терминалы оплаты и электронной записи в очередь в определенных учреждениях, разнообразные мобильные приложения и программы для рабочих целей. Программные продукты низкого качества не вызывают доверия у пользователей, и ввиду большого выбора ПО человек легко переключается на такие программные продукты, которые корректно выполняют свои функции. Соответственно, ошибки при разработке приложений играют существенную роль в притоке и оттоке пользователей таких программ. Необходимо не только проверять продукт на соответствие требованиям, но и убедиться, что он подходит для конечных пользователей и способен выполнять заявленные функции. Что же касается критичности разного рода ошибок ПО, то в определенных сферах деятельности, таких как медицина или аэрокосмическая индустрия, ошибки могут стоить жизни людей и огромных финансовых потерь компаний.

Поэтому очень важно знать и применять при разработке программ общепринятые методы верификации и оценки качества, которые позволяют вовремя выявлять и устранять ошибки на разных этапах жизненного цикла программного обеспечения, опираясь на международные стандарты. В данной статье предлагается обзор основных методов верификации ПО и методов оценки качества ПО, описывается их суть и особенности применения. Материал статьи носит обзорный характер и опирается на государственные и международные стандарты, которые мы рассмотрим далее. Целью проведения обзора является изучение основных методов верификации ПО, методов оценки качества ПО и определение современных средств автоматизации рассмотренных методов.

2. Основные понятия предметной области

2.1. Верификация программного обеспечения

Процесс разработки программного обеспечения всегда включает в себя определенное количество этапов. При этом существуют разные модели разработки ПО, в которых отличается количество и порядок прохождения этапов, что влияет на процесс тестирования. Например, в каскадной модели предполагается однократное выполнение всех этапов разработки по заданному порядку, а в итеративной определенные этапы выполняются по мере необходимости неоднократное количество раз.

В любой модели разработки ПО, как правило, в начале каждого этапа устанавливаются определенные требования, которые могут включать в себя сроки, задачи, цели. Для контроля выполнения таких требований существует заданный вид деятельности, названный верификацией. То есть это процесс оценивания компонентов разрабатываемой системы на соответствие правилам, стандартам и установленным требованиям на каждом этапе. В данном процессе также отслеживаются возможные противоречия требований к системе, проверяется корректность описания входных и выходных данных, а также интеграции компонентов ПО в систему. При этом процесс исправления возникающих противоречий и несоответствий является уже другой задачей, выходящей за рамки верификации.

В [1, с. 7] сказано, что верификация проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. Под артефактом в данном случае автор [1] понимает различные информационные сущности, документы и модели, которые создаются или используются в ходе разработки и сопровождения ПО. Например, среда разработки программного продукта также является артефактом, исходя из приведенного определения. Действительно, как сказано в [2, с. 8], в процессе верификации необходимо также проверять адекватность стандартов, процедур и среды разработки процессам жизненного цикла программных средств.

Остановимся на понятии жизненного цикла ПО. Как уже было сказано, процесс разработки ПО состоит из отдельных этапов. Под жизненным циклом ПО понимается совокупность таких этапов, которые представляют стадии процесса создания ПО. Например, в [1, с. 17] дается следующее определение. Процессом жизненного цикла ПО называется группа видов деятельности, выполняемых для решения определенного набора связанных задач по разработке или сопровождению. Таким образом, возникает понятие места верификации в жизненном цикле ПО. Процесс верификации регулирует стандарт IEEE 1012 [3].

Вернемся к понятию артефакта жизненного цикла ПО. Необходимо разделять артефакты технического характера и организационного. Например, техническое задание и исходный код программы являются техническими артефактами, а план тестирования и график работ являются организационными артефактами. Таким образом, верификация по определению должна проводиться для всех видов артефактов.

Также рассмотрим понятие валидации ПО. Как сказано в [2, с. 8], «валидация является процессом проверки соответствия системы ожиданиям заказчика». То есть этот процесс также направлен на контроль качества программного обеспечения и обнаружения в нем ошибок. Соответственно, оба понятия верификации и валидации связаны с процессами тестирования и обеспечения качества, но важно понимать, что верификация производится разработчиками, включает статический анализ и основывается на объективной оценке соответствия реализуемых компонентов, а валидация производится тестировщиками уже после верификации, включает динамический анализ и отвечает за оценку ПО в целом. Также стоит отметить, что валидация представляет собой более субъективный процесс, в котором имеет место личная оценка качества ПО представителями заказчика и различными экспертами и бизнес-аналитиками. Валидация проверяет соответствие артефактов потребностям пользователей и заказчика ПО, учитывая законы предметной области, а верификация проверяет соответствие необходимым методикам, инструментам и стандартам. Таким образом, верификация и валидация имеют общую цель, отличаясь источниками проверяемых свойств и требований.

2.2. Качество программных продуктов

Качество программного обеспечения показывает, насколько разработанный продукт отвечает требуемым свойствам. Для обеспечения качества существует определенная совокупность действий, которые проводятся над продуктом в процессе разработки, чтобы определить его актуальное состояние по вопросам готовности к выпуску, соответствия заданным требованиям и заявленному уровню качества продукта.

Качество программных средств представляет собой совокупность характеристик ПО, которые удовлетворяют определенным потребностям. Также существуют различные факторы, которые влияют на качество программного обеспечения. Например, факторы, которые связаны с полнотой и удобством, относятся к функциональным факторам. Те факторы, которые связаны с квалификацией персонала и организационной структурой, называются административными. Программно-архитектурные факторы – это те факторы, которые связаны с процессом разработки программного обеспечения.

В области оценки качества следует опираться на следующие стандарты. В России в 1992 году был утвержден и введен стандарт ГОСТ 28806-90 «Качество программных средств. Термины и определения» [4]. Данный стандарт действует и в настоящее время и содержит установленные термины и определения понятий в области качества программных средств, которые обязательны для применения во всех материалах по вычислительной технике и ПО, входящих в сферу работ по стандартизации. В 2020 году был утвержден стандарт ISO/IEC 25000:2014 [5], который содержит руководство по использованию новой серии международных стандартов оценки качества систем и программного обеспечения (SQuaRE). Данный стандарт качества предоставляет собой обзор SQuaRE, общие справочные модели и определения, взаимосвязи между документами, а также объясняет процесс перехода от старого стандарта. В 2021 году был актуализирован ГОСТ Р ИСО/МЭК 25051-2017 «Требования и оценка качества систем и программного обеспечения (SQuaRE)» [6], который является идентичным ISO/IEC 25051:2019 и содержит требования к качеству готового к использованию программного продукта и инструкции по тестированию. Также в 2021 году был актуализирован ГОСТ Р ИСО/МЭК 25010-2015 «Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов» [7], идентичный ISO/IEC 25010:2011, который определяет модель качества при использовании, в состав которой входят пять характеристик, и модель качества продукта, в состав которой входят восемь характеристик.

3. Методы верификации ПО

Рассмотрим существующие подходы к процессу верификации, изложенные в отечественной и зарубежной литературе, а также актуальные стандарты, на которые опираются методы верификации программного обеспечения. Описываемые методы разделяются на экспертизы, статический анализ, формальные методы, динамические и синтетические методы.

3.1. Экспертиза

Стандарт IEEE 1028 [8] определяет экспертизу как поверхностную проверку, нацеленную на обнаружение формальных дефектов. Экспертизы подразделяются на различные виды: техническая экспертиза, сквозной контроль, инспекция и аудит.

Существуют общие методы экспертиз, которые основаны на методе оценки по Фагану [9], и специализированные методы экспертиз, такие как организационная экспертиза и эвристическая оценка. Важным классом методов также являются систематические методы анализа архитектуры ПО. Например, систематический метод SAAM [10] фокусируется на оценке модифицируемости архитектуры и ее соответствия требованиям, которые представляются в виде сценариев.

Проверка качества заключается в том, что необходимо выполнить определенное количество операций, не связанных с компьютерными действиями. В качестве примеров можно рассмотреть действия по оценке полноты продукта, определению косметических и механических дефектов, оценку приспособленности продукта к уровню компетенций пользователей, для которых он предназначен. Также к таким операциям относится определение корректности исполнения продукта, контроль соблюдения спецификаций и требований соответствующих стандартов при исполнении, оценка комплектности продукта. Все эти действия проводятся командой экспертов под ответственностью квалифицированного руководства.

Остановимся подробнее на методе Фагана. Первоначально данный метод применялся только для проверки программного кода. Затем специалистами были сделаны выводы о том, что метод Фагана может быть использован для проверки практически любого документа. Данный подход подразумевает проведение инспекции ПО с целью получения свидетельств, которые будут подтверждать следующие факты:

1) проектная документация составлена однозначно и понятно;

2) документация по проекту взаимно согласуется;

3) документация по проекту удовлетворяет стандартам, которые были приняты организацией в соответствии с выбранной методологией жизненного цикла разработки системы.

Следуя подходу Фагана, необходимо осуществить поиск ошибок на начальных этапах разработки ПО. Это значит, что процесс должен начинаться еще во время подготовки требований к программному продукту, а также на этапе проектирования и на начальных этапах кодирования. Поиск ошибок должен быть проведен и на этапе планирования тестов. Для проведения проверки выбирают ведущего проверки, который является сторонним независимым экспертом в области разработки. Ведущий выполняет следующие функции: руководство процессом выполнения проверки, организация и ведение собраний участников, фиксирование выявленных ошибок, установка сроков подготовки отчетности, курирование процесса исправления ошибок. Также имеются роли автора, интерпретатора и оценщика.

Далее рассмотрим систематический метод SAAM. В начале описывают архитектуру ПО таким образом, чтобы можно было легко применять методы оценки по данному описанию. Затем начинается работа со сценариями. Сценарии содержат требования к ПО, и по ним составляют выборку для оценки рисков. В итоге выполнения всех перечисленных этапов происходит формирование архитектурных документов, содержащих результаты оценки. Таким образом, на каждом этапе жизненного цикла архитектуры должна формироваться соответствующая документация. Итоговая документация далее используется для обучения людей, которые будут реализовывать архитектуру.

Недостаток экспертизы заключается в том, что ее невозможно автоматизировать и в процессе ее выполнения требуется активное участие людей при верификации ПО.

3.2. Статический анализ

Статический анализ выполняется без запуска программы и позволяет проанализировать пути выполнения программы. Данный вид анализа обычно автоматизируют, это наиболее широко применяемый метод верификации. По сути, любой современный компилятор, например, GCC, предоставляет некоторый набор предупреждений о каких-либо проблемах с кодом и позволяет найти и устранить определенный спектр проблем, осуществляя простейший статический анализ. Также существуют и специальные инструменты, которые выполняют статический анализ исходного кода более детально, например, Plum Hall SQS [11], LDRA [12], Jtest [13] и многие другие. Разработчики методов данного вида анализа стремятся к надежности и полноте оценки качества кода. Но все больше шаблонов типичных ошибок переносятся в среды разработки и моделирования, такие как Qt Creator [14]. На эффективность статического анализа могут также влиять свойства языка программирования, особенности окружения, стиль программирования и прочие факторы.

Статический анализ, помимо поиска ошибок, имеет своей целью оптимизацию программного кода, обнаружение так называемого мертвого кода, а также распараллеливание и преобразование программ. Хорошим показателем считается высокая степень автоматизации данного вида анализа [15], что помогает разрабатывать высококачественный, надежный, безопасный код. Также современные разработки в области статического анализа позволяют автоматизировать верификацию свойств программного обеспечения и находить потенциально уязвимый код. При этом важно правильно определить критерии уязвимости, так как неверное их задание может приводить к возникновению большого количества ложных сигналов об ошибках в коде либо, наоборот, к пропуску реальных проблем.

3.3. Формальные методы

Формальная верификация базируется на математическом моделировании программ и требований к ПО. То есть изначально создается модель, представляющая собой идеализированное описание объекта, а затем эта модель исследуется [16]. В основе формальных методов верификации лежат логико-алгебраические, исполнимые и промежуточные формальные модели требований к ПО, поведения и окружения ПО [1].

Логико-алгебраические модели описывают некоторый набор свойств моделируемого ПО. При этом используются исчисления высказываний, исчисления предикатов и модальные логики, а также реляционные алгебры, алгебраические модели абстрактных типов данных и алгебры процессов. Построение формальных моделей не поддается автоматизации, а для того, чтобы автоматизировать анализ свойств, требуется наличие специалистов, обладающих знаниями в области математической логики и алгебры. Анализ формальных моделей включает в себя дедуктивный анализ, проверку моделей и абстрактную интерпретацию.

Формальные методы верификации способны обнаруживать сложные ошибки и часто используются в таких областях, где последствия ошибок слишком велики. На сегодняшний день особого внимания заслуживает применение формальной верификации для проверки криптографических протоколов на обеспечение специальных свойств безопасности. В этой области применяются такие средства автоматизированной верификации, как AVISPA [17], Pro Verif [18], SPIN [19] и другие.

3.4. Динамические методы

Динамические методы направлены на проверку соответствия реальной работы ПО требованиям и проектным решениям. Верификация проводится с помощью мониторинга либо

тестирования по сценариям. В процессе мониторинга происходит наблюдение и фиксация оценок работы ПО во время обычной эксплуатации. Способы мониторинга разделяются по способу получения характеристик ПО [20]: основанные на событиях и статические методы. В процессе тестирования происходит поиск ситуаций, в которых поведение программы будет некорректным, в рамках заранее подготовленных сценариев. Проводятся разные виды тестирования [21]: интеграционное тестирование, модульное, регрессионное и другие. Подготовка тестовых сценариев осуществляется вручную, а сам процесс тестирования можно легко автоматизировать [22]. При этом тестирование не доказывает полного отсутствия ошибок, а проверяет конкретные сценарии. Если в ходе такой верификации используется само ПО, то динамическая верификация называется реальной, а при использовании прототипа или исполнимой модели – имитационной. Последняя используется чаще при верификации моделей очень сложных систем. В отличие от статического анализа и формальных методов, динамические методы направлены на временные и количественные характеристики ПО и чаще применяются в таких сферах, где главным показателем является время отклика и потребление ресурсов.

3.5. Синтетические методы

Современные области синтетических методов включают в себя тестирование на основе моделей [23], которое использует формальные модели требований к ПО для построения тестов и мониторинг формальных свойств [24], основанный на внесении проверок формальных свойств ПО в систему мониторинга. Тестирование на основе моделей сегодня включает в себя методы проверки согласованности автоматов и систем переходов [25, 26, 27], методы построения тестов на основе формального анализа свойств ПО [28, 29] и методы построения тестов с помощью символического выполнения [30]. В промышленных проектах наиболее востребованы методы проверки согласованности автоматов и систем переходов.

Методы мониторинга формальных свойств ПО – это методы, использующие описание свойств с помощью обычных и временных логик [31, 32], описание свойств в виде систем переходов или автоматов [33, 34] и использующие программные контракты [35, 36, 37]. Также в настоящее время активно развивается направление синтетических методов генерации структурных тестов. Инструменты автоматической генерации тестов на основе кода, использующие статический анализ кода, формальный анализ и другие источники информации используют несколько техник разных типов [38].

Синтетические методы объединяют сильные стороны статических и динамических методов и предоставляют оптимальные решения. Так, в работе [39] предлагается синтетический метод верификации ПО, который позволяет покрыть большое количество классов ошибок и решает проблему экспоненциального роста числа состояний системы, которая встречается в формальных методах верификации при охвате всех заданных свойств проверяемой программы.

4. Методы оценки качества ПО

Далее рассмотрим основные метрики, применяемые для лексического анализа программ, а также критерии структурной сложности программ, процедурно-ориентированные и объектно-ориентированные метрики оценки качества программ.

4.1. Лексический анализ программ

При анализе текста программы с помощью подсчета количества операндов, операторов и числа их вхождений в текст программы определяется длина реализации и объем программы,

что называют лексическим анализом текста программы. Чтобы оценить характеристики программы, применяют метрики Джилба, Чепина и Холстеда [2].

Метрика Джилба предполагает число логических двоичных принятий решений в качестве меры логической сложности, которая определяет впоследствии стоимость программы на начальных этапах проектирования. При этом логическая сложность, по сути, определяется количеством условных операторов и операторов цикла в программе.

Метрика Чепина основана на оценке информационной прочности программного модуля по результатам анализа характера использования переменных, входящих в состав списка ввода и вывода.

В методе Холстеда определяются длина реализации и объем программы, а также словарь, включающий в себя число уникальных операторов и операндов программы. Затем по метрикам Холстеда выявляются недостатки программирования, такие как наличие последовательности дополняющих друг друга операторов к одному и тому же операнду, наличие неоднозначных операндов, синонимичных операндов, общих подвыражений, ненужных присваиваний и выражений, которые не представлены в свернутом виде как произведение множителей.

4.2. Структурная сложность программ

Количество взаимодействующих компонентов программы, сложность взаимодействия между ними и число связей между такими компонентами определяют структурную сложность программы. При этом создается управляющий потоковый граф, маршруты которого определяют поведение выполняющейся программы и связь между входными и выходными данными. Такой граф содержит множество вершин и дуг, а количество и сложность его маршрутов задают сложность программных модулей. Маршруты могут быть вычислительными и маршрутами принятия решений. Управляющий потоковый граф моделирует последовательность выполнения модулей и операторов программы.

Существуют определенные критерии, которые определяют структурную сложность программы. По первому критерию граф потока управления проверяется единственный раз по минимальному набору маршрутов, которые проходят через каждый оператор ветвления по каждой дуге. По второму критерию проводится анализ базовых маршрутов в программе, которые формируются и оцениваются на основе цикломатического числа графа потока управления программой [2], задающего количество проверяемых маршрутов. Также здесь используются матрицы смежности и достижимости графов, которые содержат информацию о структуре проверяемой программы. Третий критерий формирования маршрутов основан на формировании полного состава базовых структур управляющего графа и представляет собой анализ каждого из реальных ациклических маршрутов исходного графа программы и каждого цикла. При этом каждый компонент структуры программы должен анализироваться как минимум один раз [40].

Рассмотрим основные правила построения графов управления. Такой граф формируется из вершин, соответствующих операторам программы, и дуг, которые задают переходы между операторами. Таким образом учитывается логика программы и обеспечивается проведение анализа потока передачи управления между операторами. Сначала происходит учет исполнимых операторов, затем в одну вершину графа управления объединяются операторы, не изменяющие порядок действий в программе и следующие друг за другом. Следующим шагом операторы цикла заменяются несколькими вершинами. При этом в графе должны быть три группы вершин, которые задают элементы цикла – это начало цикла, тело цикла и ветвление при окончании цикла, представляющее собой завершение или возвращение к исходному оператору. Граф, построенный по таким правилам, поможет оценить структурную сложность программы, которая будет определяться количеством независимых контуров в полносвязном графе [41].

4.3. Процедурно-ориентированные метрики

Для оценки качества процедурно-ориентированных программных продуктов используются функциональные указатели и указатели свойств. Указатели показывают функциональную сложность программы, и их количество зависит от количества ссылок на внешний ввод, вывод, запрос, локальный и глобальный файл. При этом используются коэффициенты регулировки сложности, которые во многом зависят от ответов на определенные вопросы, которые связаны с влиянием различных факторов на выполнение функций ПО. Например, необходимо ответить на вопрос о влиянии скорости обработки данных, наличия средств передачи данных, простоты внесения изменений, легкости инсталляции и прочих факторов [2]. Также указывается степень влияния этих факторов. Затем происходит формирование косвенных метрик, таких как производительность, качество, удельная стоимость, документированность.

Также для оценки качества ПО используются метрики связанности модулей. Связанностью является мера прочности соединения функциональных и информационных объектов в пределах одного программного модуля. Сила связности разделяется на семь типов.

Первый тип связности, по совпадению, показывает отсутствие внутренних связей в программных модулях. Второй тип связности называется логическим и показывает связность в программных модулях, которые содержат подпрограммы для различных вариантов обращения к модулям. Третья, временная связность, характеризует связность программных модулей, составные части которых не связаны между собой, но при этом они выполняются в течение определенного периода времени. Четвертый тип, процедурная связность, показывает связность программных модулей, в которых есть порядковая связь составляющих частей, реализующих процедуру обработки данных. Пятый тип связности называется коммуникационной связностью и показывает связность программных модулей, составляющие которых используют одинаковую структуру данных. Шестой тип, информационная связность, характеризует связность программных модулей при использовании выходных данных одной части модуля как входных данных другой. И, наконец, седьмой тип называется функциональной связностью, имеет наибольшую силу и показывает такую связность, при которой модуль как единое целое реализует единственную функцию при единственной задаче.

Сцепление программных модулей также применяется при оценке качества процедурно-ориентированной программы. Сцеплением называют меру межмодульной связи, которую необходимо снижать для того, чтобы повысить качество программных средств. Таким образом, уровень качества ПО обратно пропорционален силе сцепления программных модулей [2, с. 36]. Чтобы это сцепление измерять, применяют целочисленную шкалу степеней, для каждой из которых определен тип сцепления, который можно соотнести с модулем в процессе проектирования. В первом типе сцепления по данным модуль является вызываемым, а входные параметры представлены простыми элементами данных. Второй тип сцепления называют сцеплением по образцу, он показывает сцепление, при котором модуль является вызываемым, а в качестве параметров используются агрегатные типы данных. Третий тип, сцепление по управлению, характеризует сцепление, при котором модуль является вызывающим и передает вызываемому модулю список управляющих параметров, которые сильно влияют на его работу. Следующий тип, сцепление по внешним ссылкам, показывает сцепление, при котором модуль имеет указатели на глобальный элемент данных, на который ссылается другой программный модуль. Пятый тип представляет собой сцепление по общей области и характеризует сцепление, при котором модуль совместно использует с другим модулем одну и ту же глобальную структуру данных. Последний тип, сцепление по содержанию, показывает сцепление, при котором модуль напрямую ссылается на содержимое другого модуля.

Для того, чтобы понизить степень сцепления программных модулей, необходимо удалять необязательные связи, снижать количество необходимых связей и упрощать их.

4.4. Объектно-ориентированные метрики

При оценке качества объектно-ориентированных программ применяются метрики Мартина, Чидамбера и Кемерера, Лоренца и Кидда, Абреу.

Метрики Мартина включают в себя центростремительное сцепление, центробежное сцепление, нестабильность и абстрактность. На основе этих метрик строится график, который отражает зависимость между абстрактностью и нестабильностью и называется главной последовательностью. Для обеспечения качества программного средства необходимо, чтобы классы находились как можно ближе к главной последовательности [41, 42].

Метрики Чидамбера и Кемерера представляют собой шесть метрик и основываются на анализе методов класса и дерева наследования. Первая метрика, называемая взвешенной, позволяет измерять сложность классов с учетом сложности их методов. При использовании данной метрики нужно стремиться к тому, чтобы номинальная сложность для метода была равна 1.

Вторая метрика, глубина дерева наследования, показывает самый длинный путь по иерархии классов от класса-предка к заданному классу и находит количество классов-предков, влияющих на данный класс. С увеличением глубины пути возрастает абстракция данных и снижается насыщенность класса методами. Следующая метрика называется «количество потомков» и определяет количество непосредственных потомков данного класса. С увеличением этого количества увеличивается многократность использования и снижается уровень абстракции базового класса. Связность между классами объектов, четвертая метрика, показывает количество классов, с которыми связан данный класс и характеризует статическую составляющую внешних связей классов. С увеличением связности уменьшается абстракция данных и уменьшается многократное использование класса. Количество откликов на класс находит такое количество методов, которое можно выполнить в ответ на получение сообщения данным классом. Такая метрика является мерой потенциального взаимодействия данного класса с другими классами и показывает динамику поведения объекта в системе. Шестая метрика показывает отсутствие сцепления в методах и позволяет оценить зависимость методов класса друг от друга.

Метрики Лоренца и Кидда включают в себя метрики размера, наследования, внутренние и внешние метрики. На основании определения общего количества операций и количества свойств определяется размер класса. Количество операций, переопределяемых подклассом, выявляет наличие проблем проектирования. Количество добавленных относительно родительского класса собственных методов показывает количество операций, добавленных подклассом, и может указывать на определенные трудозатраты по тестированию и внесению изменений. Индекс специализации характеризует грубую оценку степени специализации каждого подкласса при добавлении, удалении, переопределении операций. Чем выше индекс, тем выше вероятность того, что в иерархии классов имеются отдельные экземпляры, которые нарушают абстракцию суперкласса. Количество сообщений, отправляемых операцией, характеризует средний размер операции и может сигнализировать о неудачном проектировании обязанностей класса. На основе стандартных метрик сложности вычисляется сложность операции, для которой устанавливаются определенные границы. Отношение числа параметров к количеству методов класса определяет среднее количество параметров на операцию и характеризует сложность взаимодействия между объектами. Количество описаний сценариев измеряют либо количеством классов, реализующих требования к ПО, либо количеством состояний для каждого класса, либо количеством методов класса. Эта метрика позволяет эффективно оценить размер создаваемой программы. Количество ключевых классов показывает объем работы по программированию, и при превышении определенного показателя рекомендуют пересматривать выделение классов. И, наконец, количество подсистем определяется обычным подсчетом и характеризует планирование, размещение ресурсов и общие затраты на интеграцию.

Показатели качества метрик Абреу состоят из факторов закрытости методов и свойств, факторов наследования методов и свойств, фактора полиморфизма и фактора сцепления. Фактор закрытости метода характеризует процентное количество классов, из которых метод невидим. Фактор закрытости свойства показывает процентное количество классов, из которых данное свойство невидимо. Фактор наследования метода характеризует процентное количество классов, из которых метод наследован. Фактор наследования свойства характеризует процентное количество классов, из которых свойство наследовано. Фактор полиморфизма представляет отношение реального количества возможных полиморфных ситуаций к максимальному количеству возможных полиморфных ситуаций для класса. Фактор сцепления определяет наличие отношения между классами.

5. Заключение

Таким образом, в данной работе были рассмотрены базовые понятия области тестирования ПО, включающие в себя верификацию, валидацию и жизненный цикл программного обеспечения. Также определены категории факторов, влияющих на качество ПО, и произведен подробный обзор методов верификации и методов оценки качества ПО. Рассмотрены принципы экспертизы ПО, методы статического анализа, формальные, динамические и синтетические методы верификации ПО, а также перечислены актуальные средства автоматизации тестирования. Были описаны различные существующие метрики для оценки качества ПО. Особенное внимание было уделено статическому анализу как наиболее распространенному методу верификации. Также в работе рассмотрены вопросы формирования технической и пользовательской документации в разработке программ.

В статье упоминаются широко используемые стандарты, которые регулируют процессы тестирования и оценки качества ПО: стандарт IEEE 1012 [3] регулирует процесс верификации ПО и фокусируется на процессах проверки и валидации систем, программного и аппаратного обеспечения, ISO/IEC 25000:2014 [5] определяет стандарты оценки качества ПО, ГОСТ 28806-90 «Качество программных средств. Термины и определения» [4], ГОСТ Р ИСО/МЭК 25051-2017 «Требования и оценка качества систем и программного обеспечения (SQuaRE)» [6], ГОСТ Р ИСО/МЭК 25010-2015 «Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов» [7], IEEE 1028 [8] описывает методы проверки ПО.

Непрерывный рост производительности современных компьютерных систем позволяет эффективно решать все более сложные вычислительные задачи, поэтому в настоящее время в арсенале разработчиков имеется большое количество средств для разработки и тестирования качественного надежного ПО. Данная обзорная статья поможет ориентироваться в области верификации и оценки качества ПО, ее структурированный материал будет полезен для начинающих и действующих разработчиков, тестировщиков и смежных специалистов, для преподавателей вузов и студентов следующих специальностей: 09.03.01 Информатика и вычислительная техника, 09.03.02 Информационные системы и технологии, 09.03.04 Программная инженерия.

Литература

1. *Кулямин В. В.* Методы верификации программного обеспечения. М.: Институт системного программирования, 2008. 111 с.
2. *Семахин А. М.* Методы верификации оценки качества программного обеспечения: учебное пособие. Курган: Курганский государственный университет, 2018. 150 с.
3. IEEE 1012-2004 Standard for Software Verification and Validation. IEEE, 2005.

4. ГОСТ 28806-90 «Качество программных средств. Термины и определения» / Межгосударственный стандарт / Государственный комитет СССР по вычислительной технике и информатике, 1992.
5. ISO/IEC JTC 1/ SC7 Systems and software engineering – Systems and software Quality Rquirements and Evaluation (SQuaRE) – Guide to SQuaRE – Part 2: Software and systems engineering Geneva, Switzerland: ISO, 2014.
6. Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Требования к качеству готового к использованию программного продукта (RUSP) и инструкции по тестированию. Стандартинформ, 2019.
7. Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов. Стандартинформ, 2015.
8. IEEE 1028 Standard for Software Reviews. New York: IEEE, 1998.
9. *Fagan M. E.* Design and Code Inspections to Reduce Errors in Program Development // IBM Systems Journal. 1976. V. 15, № 3. P. 182–211.
10. *Kazman R., Bass L., Abowd G., Webb M. SAAM*: A Method for Analyzing the Properties of Software Architectures // Proc. 16th International Conference on Software Engineering, 1994. P. 81–90.
11. Plum Hall Inc [Электронный ресурс]. URL: `http://www.plumhall.com/` (дата обращения: 31.01.2023).
12. LDRA [Электронный ресурс]. URL: `https://ldra.com/` (дата обращения: 03.02.2023).
13. Jtest [Электронный ресурс]. URL: `https://www.parasoft.com/` (дата обращения: 3.02.2023).
14. *Шлее М.* Qt 5.10. Профессиональное программирование на C++. БХВ-Петербург, 2018.
15. PVS-Studio – статический анализатор на страже качества, защищённости (SAST) и безопасности кода [Электронный ресурс]. URL: `https://pvs-studio.com/ru/pvs-studio/` (дата обращения: 10.05.2022).
16. *Камкин А. С.* Введение в формальные методы верификации программ: учебное пособие. М.: МГУ им. М.В. Ломоносов, 2018.
17. AVISPA – Automated Validation of Internet Security Protocols and Applications [Электронный ресурс]. URL: `https://www.avispa-project.org/` (дата обращения: 03.02.2023).
18. *Бланше Б.* Моделирование и проверка протоколов безопасности с помощью прикладного исчисления Pi и проверки // Основы и тенденции в области конфиденциальности и безопасности. Октябрь 2016. № 1 (1–2). С. 1–135.
19. *Перевышина Е. А., Бабенко Л. К.* Анализ стойкости к атакам криптографических протоколов с использованием формального верификатора SPIN // Известия ЮФУ. Технические науки. 2019. № 5 (207). С. 58–68.
20. *Гурин Р. Е., Рудаков И. В., Ребриков А. В.* Методы верификации программного обеспечения // Сетевое научное издание. Наука и образование. 2015. № 10. С. 235–251.
21. *Пероцкая В. Н., Градусов Д. А.* Основы тестирования программного обеспечения: учебное пособие. Владимир: Издательство ВЛГУ, 2017.
22. Топ 10 бесплатных инструментов для автоматизированного тестирования [Электронный ресурс]. URL: `https://medium.com/nuances-of-programming/топ-10-бесплатных-автоматизированных-инструментов-для-тестирования-32e461a78298` (дата обращения: 03.02.2023).
23. *Utting M., Pretschner A., Legeard B.* A Taxonomy of Model-Based Testing. Technical Report, Department of Computer Science, The University of Waikato, New Zealand, 2006.
24. *Delgado N., Gates A. Q., Roach S.* A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools // IEEE Transactions on Software Engineering. December 2004. V. 30, № 12. P. 859–872.

25. *Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A.* Model Based Testing of Reactive Systems. LNCS 3472, Springer, 2005.
26. *Utting M., Legeard B.* Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann, 2007.
27. *Ambert F., Bouquet F., Chemin S., Guenaud S., Legeard B., Peureux F., Vacelet N., Utting M.* BZ-TT: A tool-set for test generation from Z and B using constraint logic programming // Proc. FATES'2002, August 2002. P. 105–119.
28. *Ammann P., Black P. E.* Abstracting formal specifications to generate software tests via model checking // Proc. 18th IEEE Digital Avionics Systems Conference, October 1999.
29. *Engel C., Hahnle R.* Generating unit tests from formal proofs // Proc. TAP 2007. LNCS 4454, Springer-Verlag, 2007. P. 169–188.
30. *Sen K., Agha G.* CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools // Proc. Computer Aided Verification, August 2006. P. 419–423.
31. *Lee I., Kannan S., Kim M., Sokolsky O., Viswanathan M.* Runtime Assurance Based On Formal Specifications // Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 1999. P. 279–287.
32. *Havelund K., Rosu G.* Monitoring Java Programs with Java PathExplorer // Electronic Notes in Theoretical Computer Science. January 2004. V. 55, № 2. P. 1–18.
33. *Cavalli A., Gervy C., Prokopenko S.* New approaches for passive testing using an Extended Finite State Machine specification // Information and Software Technology. Elsevier. September 2003. V. 45, № 12. P. 837–852.
34. *Chen D., Wu J. Chi H.* Passive testing on TCP // Proc. International Conference on Communication Technology (ICCT), April 2003. V. 1. P. 182–186.
35. *Cheon Y., Leavens G. T.* A runtime assertion checker for the Java Modeling Language (JML) // Proc. International Conference on Software Engineering Research and Practice (SERP'02), CSREA Press, June 2002. P. 322–328.
36. *Schoeller B.* Strengthening Eiffel Contracts using Models // Proc. Workshop on Formal Aspects of Component Software (FACS), September 2003. P. 143–158.
37. *Barnett M., Schulte W.* Runtime verification of .NET contracts // Journal of Systems and Software. March 2003. V. 65, № 3. P. 199–208.
38. *Кларк Э. М., Гамбург О., Пелед Д.* Верификация моделей программ: Model Checking: пер. с англ. / под ред. Р. Смелянского. М.: МЦНМО, 2002. 416 с.
39. *Рудаков И. В.* Разработка и исследование синтетического метода верификации программы с помощью SMT-решателей // Вестник Московского государственного технического университета им. Н. Э. Баумана. Серия Приборостроение. 2016. № 4 (109). С. 49–64.
40. *Черников Б. В.* Управление качеством программного обеспечения: учебник. М.: Форум, 2012. 240 с.
41. *Черников Б. В., Поклонов Б. Е.* Оценка качества программного обеспечения: Практикум: учебное пособие. М.: ФОРУМ: ИНФРА-М, 2012. 400 с.
42. *Сандлер К., Баджет Т., Майерс Г.* Искусство тестирования программ. М.: Вильямс, 2015. 272 с.

Мерзлякова Екатерина Юрьевна

к.т.н., доцент кафедры прикладной математики и кибернетики, Сибирский государственный университет телекоммуникаций и информатики (СибГУТИ, 630102, Новосибирск, ул. Кирова, д. 86), тел. +7 383 2698 272, e-mail: `katerina.artist@yandex.ru`, ORCID ID: 0000-0001-6847-5588.

Янченко Елена Викторовна
к.т.н., доцент кафедры прикладной математики и кибернетики, Сибирский государственный университет телекоммуникаций и информатики (СибГУТИ, 630102, Новосибирск, ул. Кирова, 86), тел.+7 383 2698 272, e-mail: eku06@mail.ru, ORCID 0009-0004-4851-9837.

Авторы прочитали и одобрили окончательный вариант рукописи.
Авторы заявляют об отсутствии конфликта интересов.
Вклад соавторов: *Каждый автор внес равную долю участия как во все этапы проводимого теоретического исследования, так и при написании разделов данной статьи.*

Review of Software Quality Verification and Evaluation Methods

Ekaterina Y. Merzlyakova, Elena V. Yanchenko

Siberian State University of Telecommunications and Information Science (SibSUTIS)

Abstract: The basic concepts in the field of software quality testing and evaluation are defined. In particular, the concepts of verification, validation and software lifecycle and their role in the testing process are considered. The concept of an artifact in the software lifecycle is considered. The Russian and international standards in the field of software quality assessment are taken into account. The existing methods of software verification are regarded: expertise, static analysis, formal methods, dynamic and synthetic methods.
An overview of software quality assessment methods used in software development and testing cycles is presented. Functional and administrative factors affecting software quality are also considered. The article lists modern testing automation tools that can be used today. The review material of the article will be useful for software developers, novice testers and related specialists.

Keywords: software testing, verification, validation, program analysis, software lifecycle, testing automation, quality assessment, software verification methods, software development, software quality.

For citation: Merzlyakova E. Y., Yanchenko E. V. Review of software quality verification and evaluation methods (in Russian). *Vestnik SibGUTI*, 2023, vol. 17, no. 1, pp. 92-106. https://doi.org/10.55648/1998-6920-2023-17-1-92-106.

The article was submitted: 17.10.2022;
revised version: 30.01.2023;
accepted for publication 10.02.2023.

References

1. Kulyamin V. V. *Metody verifikatsii programmnogo obespecheniya* [Software verification methods]. Moscow, Institut sistemnogo programmirovaniya, 2008, 111 p.
2. Semakhin A. M. *Metody verifikatsii otsenki kachestva programmnogo obespecheniya* [Software quality assessment verification methods]. Kurganskii gosudarstvennyi universitet, Kurgan, 2018. 150 p.
3. IEEE 1012-2004 Standard for Software Verification and Validation. IEEE, 2005.

4. GOST 28806-90 *Kachestvo programmnykh sredstv. Terminy i opredeleniya* [Quality of software. Terms and Definitions]. Gosudarstvennyy komitet SSSR po vychislitel'noy tekhnike i informatike, 1992.
5. ISO/IEC JTC 1/ SC7 Systems and software engineering - Systems and software Quality Rquirements and Evaluation (SQuaRE) - Guide to SQuaRE - Part 2: Software and systems engineering Geneva, Switzerland: ISO, 2014.
6. *Informatsionnyye tekhnologii. Sistemnaya i programmnaya inzheneriya. Trebovaniya i otsenka kachestva sistem i programmnogo obespecheniya (SQuaRE). Trebovaniya k kachestvu gotovogo k ispol'zovaniyu programmnogo produkta (RUSP) i instruktsii po testirovaniyu* [Information technology. System and software engineering. Requirements and quality assessment of systems and software (SQuaRE). Quality requirements for a ready-to-use software product (RUSP) and testing instructions]. Federal'noye agentstvo po tekhnicheskomu regulirovaniyu i metrologii, Standartinform, 2019.
7. *Informatsionnyye tekhnologii. Sistemnaya i programmnaya inzheneriya. Trebovaniya i otsenka kachestva sistem i programmnogo obespecheniya (SQuaRE). Modeli kachestva sistem i programmnykh produktov* [Information technology. System and software engineering. Requirements and quality assessment of systems and software (SQuaRE). Quality models of systems and software products]. Federal'noye agentstvo po tekhnicheskomu regulirovaniyu i metrologii, Standartinform, 2019.
8. IEEE 1028 Standard for Software Reviews. New York: IEEE, 1998.
9. Fagan M. E. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 1976, vol. 15, no. 3, pp. 182-211.
10. Kazman R., Bass L., Abowd G., Webb M. SAAM: A Method for Analyzing the Properties of Software Architectures. *Proc. of 16-th International Conference on Software Engineering*, 1994, pp. 81-90.
11. Plum Hall Inc, available at: `http://www.plumhall.com/` (accessed 31.01.2023).
12. LDRA, available at: `https://ldra.com/` (accessed 3.02.2023).
13. Jtest, available at: `https://www.parasoft.com/` (accessed 3.02.2023).
14. *Shleye M. Qt 5.10. Professional'noye programmirovaniye na C++* [Qt 5.10. Professional programming in C++]. BKHV-Peterburg, 2018.
15. *PVS Studio – staticheskiy analizator na strazhe kachestva, zashchishchonnosti (SAST) i bezopasnosti koda* [PVS Studio - a static analyzer guarding the quality, security (SAST) and security of the code], available at: `https://pvs-studio.com/ru/pvs-studio/` (accessed 10.05.2022).
16. *Kamkin A. S. Vvedeniye v formal'nyye metody verifikatsii programm* [Introduction to formal methods of program verification]. Moskva, MGU imeni M.V. Lomonosov, 2018.
17. AVISPA - Automated Validation of Internet Security Protocols and Applications, available at: `https://www.avispa-project.org/` (accessed 3.02.2023).
18. *Blanshe B. Modelirovaniye i proverka protokolov bezopasnosti s pomoshch'yu prikladnogo ischisleniya Pi i proverki* [Modeling and verification of security protocols using applied calculus Pi and verification] *Osnovy i tendentsii v oblasti konfidentsial'nosti i bez-opasnosti. Oktyabr' 2016, no.1(1-2). pp.1-135.*
19. *Perevyshina, Ye. A., Babenko L. K. Analiz stoykosti k atakam kriptograficheskikh protokolov s ispol'zovaniyem formal'nogo verifikatora SPIN* [Analysis of resistance to attacks of cryptographic protocols using the formal verifier SPIN]. *Izvestiya YUFU. Tekhnicheskiye nauki. 2019, no. 5(207). pp. 58-68.*
20. *Gurin R.Ye., Rudakov I.V., Rebrikov A.V. Metody verifikatsii programmnogo obespecheniya* [Software verification methods]. *Setevoye nauchnoye izdaniye. Nauka i obrazovaniye. MGTU im. N.E. Baumena. Elektronnyy zhurnal no. 10, 2015.* pp.235-251.
21. *Perotskaya V. N., Gradusov D. A. Osnovy testirovaniya programmnogo obespecheniya* [Fundamentals of software testing]. Vladimir, Izdatel'stvo VLGU, 2017.
22. *Top 10 besplatnykh instrumentov dlya avtomatizirovannogo testirovaniya* [Top 10 Free Automated Testing Tools], available at: `https://medium.com/nuances-of-programming/топ-10-бесплатных-автоматизированных-инструментов-для-тестирования-32e461a78298` (accessed 3.02.2023).
23. Utting M., Pretschner A., Legeard B. *A Taxonomy of Model-Based Testing*. Technical Report, Department of Computer Science, The University of Waikato, New Zealand, 2006.
24. Delgado N., Gates A. Q., Roach S. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools. *IEEE Transactions on Software Engineering*, 2004, vol. 30, no. 12, pp. 859-872.
25. Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A. (eds.). *Model Based Testing of Reactive Systems*. LNCS 3472, Springer, 2005.
26. Utting M., Legeard B. *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann, 2007.

27. Ambert F., Bouquet F., Chemin S., Guenaud S., Legeard B., Peureux F., Vacelet N., Utting M. BZ-TT: A tool-set for test generation from Z and B using constraint logic programming. *Proc. of FATES'2002*, 2002, pp. 105-119.
28. Ammann P., Black P. E. Abstracting formal specifications to generate software tests via model checking. *Proc. of 18-th Digital Avionics Systems Conference*, 1999, vol. 2, p. 10.
29. Engel C., Hahnle R. Generating unit tests from formal proofs. Y. Gurevich, B. Meyer, eds. *Proc. of TAP 2007*. LNCS 4454, Springer-Verlag, 2007. pp. 169–188.
30. Sen K., Agha G. CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools. *Proc. of Computer Aided Verification*, 2006, pp.419-423.
31. Lee I., Kannan S., Kim M., Sokolsky O., Viswanathan M. Runtime Assurance Based On Formal Specifications. *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'1999*, 1999, pp. 279-287.
32. Havelund K., Rosu G. Monitoring Java Programs with Java PathExplorer. *Electronic Notes in Theoretical Computer Science*, 2004, vol. 55, no. 2, pp. 1-18.
33. Cavalli A., Gervy C., Prokopenko S. New approaches for passive testing using an Extended Finite State Machine specification. *Information and Software Technology*, 2003, vol. 45, no. 12, pp. 837-852.
34. Chen D., Wu J. Chi H. Passive testing on TCP. *Proc. of International Conference on Communication Technology ICCT 2003*, 2003, vol. 1, pp. 182-186.
35. Cheon Y., Leavens G. T. A runtime assertion checker for the Java Modeling Language (JML). *Proc. of International Conference on Software Engineering Research and Practice (SERP'02)*, 2002, pp. 322-328.
36. Schoeller B. Strengthening Eiffel Contracts using Models. *Proc. of Workshop on Formal Aspects of Component Software FACS'03*, 2003, pp. 143-158.
37. Barnett M., Schulte W. Runtime verification of .NET contracts. *Journal of Systems and Software*, 2003, vol. 65, no. 3, pp. 199-208.
38. Clark E.M., Humburg O., Peled D. *Verifikatsiya modelei programm: Model Checking* [Verification of program models: Model Checking]. Moscow center for continuous mathematical education, Moscow, 2002. 416 p.
39. Rudakov I.V. *Razrabotka i issledovanie sinteticheskogo metoda verifikatsii programmy s pomoshch'yu SMT-reshatelei* [Development and research of a synthetic method for program verification using SMT solvers], *Vestnik MGTU im. N.E. Baumana, Seriya Priborostroyeniye*, 2016, no. 4(109), pp. 49-64.
40. Chernikov B.V. *Upravlenie kachestvom programmnogo obespecheniya* [Software quality management]. Moscow, Forum, 2012. 240 p.
41. Chernikov B.V., Poklonov B.E. *Otsenka kachestva programmnogo obespecheniya* [Software quality assessment]. Moscow, Forum:Infra-M, 2012. 400 p.
42. Sandler K., Budget T., Mayers G. *Iskusstvo testirovaniya program* [The art of software testing]. Moscow, Williams, 2015. 272 p.

Ekaterina Y.Merzlyakova

PhD (Engineering), Docent of the Department of Applied Mathematics and Cybernetics, Siberian State University of Telecommunications and Informatition Science (SibGUTI, 630102, Novosibirsk, Kirova str., 86), tel. +7 383 2698 272, e-mail: katerina.artist@yandex.ru, ORCID ID: 0000-0001-6847-5588.

Elena V.Yanchenko

PhD (Engineering), Docent of the Department of Applied Mathematics and Cybernetics, Siberian State University of Telecommunications and Information Science (SibGUTI, 630102, Novosibirsk, Kirova str., 86) tel.+7 383 2698 272, e-mail: eku06@mail.ru, ORCID 0009-0004-4851-9837.