Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Оренбургский государственный университет»

Кафедра программного обеспечения вычислительной
техники и автоматизированных систем

Т.М. Зубкова

ВЕРИФИКАЦИЯ МОДЕЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания

Рекомендовано к изданию редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательным программам высшего образования по направлениям подготовки 09.04.04 Программная инженерия, 09.04.01 Информатика и вычислительная техника

Оренбург

2019

УДК 681.3 (07)
ББК 32.973.26-018я73
З 91

Рецензент – профессор, доктор технических наук Н.А. Соловьев

Зубкова, Т.М.
З 91 Верификация моделей программного обеспечения: методические указания /Т.М. Зубкова; Оренбургский гос. ун-т.- Оренбург: ОГУ, 2019. – 50 с.

Методические указания для выполнения практических и лабораторных работ по дисциплине «Методы верификации моделей программного обеспечения» предназначены для оказания помощи студентам при выполнении индивидуальных заданий. Данная дисциплина входит в вариативную часть обязательных дисциплин магистрантов очной формы обучения по направлению подготовки 09.04.04 Программная инженерия с профилем подготовки «Разработка программно-информационных систем» и 09.04.01 Информатика и вычислительная техника с профилем подготовки «Информационное и программное обеспечение автоматизированных систем» по типу образовательной программы «Программа академической магистратуры».

В методических указаниях изложены задания, теоретические основы для их выполнения.

УДК 681.3 (07)
ББК 32.973.26-018я73

Содержание

Введение

Цель освоения дисциплины: ознакомить и обучить основным методам и подходам верификации программного обеспечения.

Задачи:

- обучить понятиям верификации ПО в рамках жизненного цикла;
- обучить видам экспертиз верификации ПО;
- обучить формальным методам верификации ПО;
- обучить динамическим методам верификации ПО;
- обучить синтетическим методам верификации ПО.

В результате изучения дисциплины студенты должны знать, уметь и владеть пониманием существующих подходов к верификации моделей программного обеспечения.

Процесс изучения дисциплины направлен на формирование следующих результатов обучения.

Знать: существующие подходы к верификации моделей программного обеспечения.

Уметь: осуществлять верификацию ПО.

Владеть: навыками понимания существующих подходов к верификации моделей ПО.

1 Теоретические основы дисциплины «Методы верификации моделей программного обеспечения»

1.1 Основные понятия

Для обеспечения корректности и надежности работы программных систем большое значение имеют различные методы верификации и валидации, позволяющие выявлять ошибки на разных этапах разработки и сопровождения ПО, чтобы последовательно устранять их.

Верификация обозначает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или, что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.

Валидация – это проверка того, что сам продукт правилен, т.е. подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

Жизненный цикл программного обеспечения это весь интервал времени от момента зарождения идеи о том, чтобы создать или приобрести программную систему для решения определенных задач, до момента полного прекращения использования последней ее версии.

Описать общую структуру жизненного цикла произвольной программной системы, довольно сложно – слишком сильно отличается разработка и развитие ПО, предназначенного для решения разных задач в различных окружениях. Однако можно определить набор понятий, в терминах которых описывается любая такая структура – это, прежде всего, виды деятельности, роли и артефакты

Вид деятельности в жизненном цикле ПО – это набор действий, направленных на решение одной задачи или группы тесно связанных задач в рамках разработки и сопровождения ПО. Примерами видов деятельности являются анализ предметной области, выделение и описание требований, проектирова-

ние, разработка кода, тестирование, управление конфигурациями, развертывание.

Роль в жизненном цикле ПО – это профессиональная специализация людей, участвующих в работах по созданию или сопровождению ПО и имеющих одинаковые интересы или решающих одни и те же задачи по отношению к этому ПО. Примеры ролей: бизнес-аналитик, инженер по требованиям, архитектор, проектировщик пользовательского интерфейса, программист-кодировщик, технический писатель, тестировщик, руководитель проекта, пользователь, администратор системы.

Артефактами жизненного цикла ПО называются различные информационные сущности, документы и модели, создаваемые или используемые в ходе разработки и сопровождения ПО. Так, артефактами являются техническое задание, описание архитектуры, модель предметной области на каком-либо графическом языке, исходный код, пользовательская документация и т.д. Различные модели, используемые отдельными разработчиками при создании и анализе ПО, но не зафиксированные в виде доступных другим людям документов, не могут считаться артефактами.

Верификация и валидация являются видами деятельности, направленными на контроль качества программного обеспечения и обнаружение ошибок в нем. Имея общую цель, они отличаются источниками проверяемых в их ходе свойств, правил и ограничений, нарушение которых считается ошибкой.

Верификация проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО.

Кроме того, проверяется, что требования, проектные решения, документация и код оформлены в соответствии с нормами и стандартами, принятыми в

данной стране, отрасли и организации при разработке ПО, а также – что при их создании выполнялись все указанные в стандартах операции, в нужной последовательности.

Валидация проверяет соответствие любых создаваемых или используемых в ходе разработки и сопровождения ПО артефактов нуждам и потребностям пользователей и заказчиков этого ПО, с учетом законов предметной области и ограничений контекста использования ПО. Эти нужды и потребности чаще всего не зафиксированы документально – при фиксации они превращаются в описание требований, один из артефактов процесса разработки ПО. Различие между верификацией и валидацией проиллюстрировано на рисунке 1.

Рисунок 1 – Соотношение верификации и валидации

1.2 Место верификации в жизненном цикле ПО

Хотя общую структуру жизненного цикла произвольной программной системы определить невозможно, существует несколько наиболее часто используемых способов организации различных видов деятельности в рамках жизненного цикла. Их называют моделями жизненного цикла ПО.

1.2.1 Задачи верификации в рамках жизненного цикла ПО

Все используемые на практике модели жизненного цикла по схеме организации работ являются разновидностями либо каскадной, либо итеративной модели, поэтому независимо от процесса разработки ПО верификация играет в нем ключевую роль, решая следующие задачи:

- выявление дефектов (ошибок, недоработок, неполноты и пр.) различных артефактов разработки ПО (требований, проектных решений, документации или кода), что позволяет устранять их и поставлять пользователям и заказчикам более правильное и надежное ПО;
- выявление наиболее критичных и наиболее подверженных ошибкам частей создаваемой или сопровождаемой системы;
- контроль и оценка качества ПО во всех его аспектах;
- предоставление всем заинтересованным лицам (руководителям, заказчикам, пользователям и пр.) информации о текущем состоянии проекта и характеристиках его результатов;
- предоставление руководству проекта и разработчикам информации для планирования дальнейших работ, а также для принятия решений о продолжении проекта, его прекращении или передаче результатов заказчику.

1.2.2 Верификация и другие процессы разработки и сопровождения ПО

Процессом жизненного цикла ПО называется группа видов деятельности, выполняемых для решения определенного набора связанных задач по разработке или сопровождению ПО. На сегодняшний день нет четко определенного общего списка процессов, который не вызывал бы возражений у тех или иных исследователей и практиков. Международные стандарты ISO 12207, IEEE 1074 , ISO 15288 , ISO 15504 используют несколько отличающиеся системы процессов (все на процессы ЖЦ ПО).

По ISO 12207 к верификации имеют отношение 5 процессов:

- обеспечение качества (quality assurance),
- собственно верификация,
- валидация,
- совместные экспертизы (joint review)
- и аудит (audit).

Тестирование целиком отнесено к валидации. Кроме того, выделен процесс разрешения проблем (problem resolution), для которого верификация и валидация поставляют входные данные (те самые проблемы).

IEEE 1074 выделяет только один связанный с верификацией процесс – группу деятельностей по оценке (evaluation), которая включает экспертизы (review), аудиты, прослеживание требований и тестирование. Еще несколько видов деятельности, которые можно отнести к верификации и валидации, разбросаны по другим процессам – сбор и анализ метрик, анализ осуществимости, определение потребностей в улучшении ПО, валидация программы обучения.

ISO 15288 считает отдельными процессами управление качеством, оценивание, верификацию и валидацию.

В ISO 15504 (SPICE) в качестве процессов выделены совместные экспертизы и аудиты (один процесс), управление качеством, обеспечение качества и экспертизы (review). Тестирование считается частью других процессов – реализации и интеграции ПО и интеграции программно-аппаратной системы в целом.

С технической точки зрения верификация и валидация являются неотъемлемыми элементами деятельности по обеспечению качества. С одной стороны, эта деятельность должна обеспечить формирование критериев качества, использование при разработке доказавших свою эффективность технологий, определение и контроль процедур выполнения отдельных операций, точность, согласованность и полноту при описании требований, проектных решений, пользовательской документации, формулировку требований и проектных решений на необходимом уровне абстракции. С другой стороны, в рамках обеспечения

качества с помощью верификации и валидации необходимо оценивать текущие характеристики качества ПО и отдельных артефактов процесса разработки и сопоставлять их с критериями и правилами, определенными в рамках системы обеспечения качества проекта и организации в целом. Деятельность по управлению качеством отличается от обеспечения качества, по-видимому, только большим акцентом на административных процедурах и предварительном определении целей обеспечения качества, на прямой ответственности руководства проекта за качество его результатов.

Экспертизы и аудит, в свою очередь, являются методами проведения верификации и валидации, такими же, как тестирование, оценка архитектуры на основе сценариев или проверка моделей. В стандартах они рассматриваются как отдельные процессы, скорее всего, потому что применимы к произвольным артефактам жизненного цикла в рамках любого вида деятельности, а также часто используются для оценки процессов и организационных видов деятельности в проекте, в отличие от большинства других методов верификации.

1.2.3 Верификация различных артефактов жизненного цикла ПО

Артефакты жизненного цикла ПО делят на технические и организационные.

К техническим артефактам относятся описание требований (техническое задание), описание проектных решений (эскизный и технический проекты), исходный код (текст программы), документация пользователей и администраторов (рабочая документация), сама работающая система. Техническими также являются вспомогательные артефакты для проведения верификации и валидации – формальные модели требований и проектных решений, наборы тестов и компоненты тестового окружения, модели поведения реального окружения системы.

Организационными артефактами являются структура работ, разнообразные проектные планы (план-график работ, план конфигурационного управле-

ния, план обеспечения качества, план обхода и преодоления рисков, планы проверок и испытаний и пр.), описания системы качества, описания процессов и процедур выполнения отдельных работ. Верификация может и должна проводиться для всех видов артефактов, создаваемых при разработке и сопровождении программных систем.

При верификации организационных документов и процессов проверяется, насколько выбранные формы организации, планы и методы выполнения работ соответствуют задачам, решаемым в рамках проекта, и ограничениям по срокам и бюджету, то есть, что с помощью выбранных методов и технологий проект действительно можно выполнить в рамках контракта. Проверяется также, что команда проекта в достаточной степени владеет используемыми технологиями разработки, или же, что запланированы необходимые мероприятия по обучению.

При верификации описания требований одной из первых задач верификации является оценка осуществимости требований с помощью технологий, взятых на вооружение в проекте и в рамках выделенных на проект ресурсов. Проверяются также характеристики требований, указанные в стандартах IEEE 830 и IEEE 1233, а именно следующие:

- однозначность. Требования должны однозначно, недвусмысленно выражать нужные ограничения;
- непротиворечивость или согласованность. Различные требования не должны противоречить друг другу или основным законам предметной области;
- внутренняя полнота. Требования должны описывать поведения системы во всех возможных в контексте ее работы ситуациях. Все значимые законы предметной области и нормы действующих в ней стандартов должны быть учтены в требованиях как ограничения на работу системы;
- минимальность. Требования не должны быть сводимы друг к другу на основе формальной логики и основных законов предметной области;

- проверяемость. В каждой затрагиваемой требованием ситуации должен быть способ однозначно установить, выполнено оно или нарушено;
- систематичность. Требования должны быть представлены в рамках единой системы, с четким указанием связей между ними, с уникальными идентификаторами и набором определенных атрибутов: приоритетом, риском внесения изменений, критичностью для пользователей и пр.

Кроме этого, требования должны адекватно и достаточно полно отражать нужды и потребности пользователей и других заинтересованных лиц. Для проверки адекватности и полноты отражения реальных потребностей пользователей необходимо проводить валидацию.

При верификации проектных решений проверяются следующие свойства:

- все проектные решения связаны с требованиями и действительно нацелены на их реализацию. Все требования нашли отражение в проектных решениях;
- проектные документы точно и полно формулируют принятые решения, отдельные их элементы не противоречат друг другу;
- при оформлении проектных документов учтены все правила корректности составления документов такого типа на соответствующих языках. Если используются графические нотации, такие как DFD, ERD или UML, то все диаграммы составлены с соблюдением всех правил и ограничений этих языков;
- для проектных решений, связанных с критически важными требованиями к системе, например, по ее безопасности и защищенности, необходимо с помощью максимально строгих методов установить их корректность, т.е. то, что они действительно реализуют соответствующие требования во всех возможных в контексте работы системы ситуациях.

При верификации исходного кода системы проверяют указанные ниже характеристики:

- все элементы кода связаны с проектными решениями и требованиями и корректно реализуют соответствующие проектные решения;

- код написан в соответствии с синтаксическими и семантическими правилами выбранных языков программирования, а также с принятыми в организации и данном проекте стандартами оформления текстов программ (coding rules, coding conventions). Выполнены требования к удобству сопровождения кода;
- в исходном коде отсутствуют пути выполнения, достижимые в условиях работы системы и приводящие к ее сбоям, зацикливаниям или тупиковым ситуациям, разрушению процессов и данных проверяемой системы или объемлющей, исключительным ситуациям, непредусмотренным в требованиях и проектных решениях, и пр.

Верификация самой работающей системы или ее компонентов, которые можно выполнять независимо, призвана проверить следующее:

- система или ее компоненты действительно способны работать в том окружении, в котором они нужны пользователям, или же в рамках достаточно точной имитации этого окружения;
- поведение системы или ее компонентов на возможных сценариях их использования соответствует требованиям по всем измеримым характеристикам. Это, снова, невозможно проверить полностью. Однако, для наиболее критичных требований и сценариев использования применяются более строгие и полные методы проверки соответствия.

Часто проверяется также соответствие поведения системы и ее компонентов реальным нуждам пользователей – это уже является валидацией.

При верификации пользовательской документации проверяется следующее:

- документация содержит полное, точное и непротиворечивое описание поведения системы;
- описанное в документации поведение соответствует реальному поведению системы.

Верификации также должны подвергаться тестовые планы или планы других мероприятий по верификации, а также тесты или материалы, подготовленные для проведения верификации других артефактов, например различные формальные модели. В этих случаях проверяются такие характеристики:

- подготовленные планы соответствуют основным рискам проекта и уделяют различным его артефактам ровно такое внимание, которое требуется, исходя из их зрелости и важности для проекта;
- методы верификации, которые планируется применять, действительно способны дать лучшие результаты (с точки зрения обнаружения ошибок и получения достоверных оценок качества, отнесенных к затратам) в намеченных для них областях;
- подготовленные материалы (тесты, списки возможных ошибок для инспекций, формальные модели требований или окружения системы и пр.) соответствуют контексту использования системы, требованиям к проверяемым артефактам и связанным с ними проектным решениям и могут быть использованы в качестве входных данных для выбранных методов проведения верификации.

1.2.4 Международные стандарты, касающиеся верификации ПО

Основным стандартом, регулирующим планирование и проведение верификации ПО, является стандарт IEEE 1012 на процессы верификации и валидации. Этот стандарт содержит следующую информацию.

Описание наборов отдельных задач верификации, соответствующих разным видам деятельности в рамках процессов, определенных в ISO 12207. Эти задачи выделены таким образом, чтобы как можно детальнее и полнее оценить результаты соответствующих видов деятельности:

- подготовка планов проведения верификации и оценка их соответствия требованиям к ПО, ресурсам проекта и его рискам, а также используемым технологиям;
- оценка всех основных артефактов жизненного цикла – концепции системы, описания требований, проектных решений, кода и документации – в соответствии с описанными выше критериями;
- анализ критичности – определение уровня критичности отдельных требований, проектных решений, модулей, элементов кода и документации, связанных с ними рисков, их важности для достижений целей проекта, безопасности и экономической эффективности разрабатываемой или сопровождаемой системы;
- анализ привязки требований и их прослеживание – определение связей между требованиями, проектными решениями, отдельными модулями и функциями, тестами, разделами документации;
- анализ интерфейсов – оценка корректности интерфейсов модулей и отдельных функций с точки зрения полноты, адекватности и точности отражения реализованных в них требований и проектных решений;
- анализ возможных сбоев – выделение наиболее рискованных и критичных ситуаций при работе ПО с точки зрения безопасности и экономической эффективности его эксплуатации, оценка возможного ущерба от них и их вероятности;
- анализ защищенности – анализ возможностей несанкционированного доступа к данным, коммуникациям и исполняемым модулям системы, возможностей получения контроля над ее работой в различных аспектах, а также оценка связанных с этим рисков;
- подготовка вспомогательных артефактов для верификации, прежде всего, разработка системных, интеграционных и модульных тестов;
- выполнение тестов всех уровней и анализ их результатов.

Рекомендуемый шаблон плана проведения верификации и валидации, с описанием содержания основных его разделов. Этот план предполагает технически обоснованный выбор методов верификации в начале проекта, большая часть его пунктов просто соответствует выделенным в стандарте задачам верификации. Для помощи при создании планов проведения верификации и валидации ранее был принят стандарт IEEE 1059, но его содержание полностью покрывается IEEE 1012.

Определение 4-х уровней критичности ПО (software integrity levels), от высокой (сбой в таких системах способен привести к невосполнимому ущербу – потери человеческой жизни, разрушению инфраструктуры, большим экономическим потерям) до минимальной (сбои имеют пренебрежимые последствия). Для каждого уровня определен минимальный набор задач верификации, которые рекомендуется выполнять для такого ПО.

Отдельные методы и задачи верификации и валидации описываются в следующих стандартах.

IEEE 829 – документация тестирования ПО. Это базовый стандарт, описывающий вспомогательные артефакты для тестирования. Основными такими артефактами являются следующие:

- *тестовый план* (test plan) – основной документ, связывающий разработку тестов и тестирование с задачами проекта;
- *тестовые варианты* (test case) – сценарии проведения отдельных тестов. Каждый тестовый вариант предназначен для проверки определенных свойств некоторых компонентов системы в определенной конфигурации и включает:
  - действия по инициализации тестируемой системы (или компонента);
  - приведение ее в необходимое состояние, вместе с предыдущим шагом этот составляет *преамбулу* тестового варианта;

- набор воздействий на систему, выполнение которых должно быть проверено данным тестом;
- действия по проверке корректности поведения системы – сравнение полученных и ожидаемых результатов, проверка необходимых свойств результатов, проверка инвариантов данных системы и пр.;
- финализацию системы – освобождение захваченных при выполнении теста ресурсов.

- *Описания тестовых процедур* (test procedure specifications). Тестовые процедуры могут быть представлены в виде скриптов или программ, автоматизирующих запуск тестовых вариантов, или в виде инструкций для человека, следуя которым можно выполнить те же варианты.
- *Отчеты о нарушениях* (test incident reports) – детальное описание отдельных ошибок, обнаруженных при выполнении тестов, с указанием всех условий, необходимых для их проявления, нарушаемых требований и ограничений, возможных последствий, а также предварительной локализации ошибки в одном или нескольких модулях.
- *Итоговый отчет о тестировании* (test summary report) – отчет с суммарной информацией о результатах тестов, включающей достигнутое тестовое покрытие по используемым критериям и общую оценку качества компонентов системы, проверяемых тестами.

Модульное тестирование ПО регулируется стандартом IEEE 1008, не пересматривавшимся с 1987 года. Этот стандарт достаточно детально описывает процедуру подготовки модульных тестов, их выполнения и оценки результатов, состоящую из 8-ми следующих видов деятельности:

- планирование, включающее в себя определение используемых методов тестирования, критериев полноты тестов и критерия окончания тестирования, а также определение необходимых ресурсов;
- определение проверяемых требований и ограничений;
- уточнение и детализация планов;

- разработка набора тестовых вариантов;
- выполнение тестов;
- проверка достижения критерия окончания тестирования и оценка полноты тестов по выбранным критериям;
- оценка затрат ресурсов и качества протестированных модулей.

Группа стандартов ISO 14598 описывают процессы оценки программных систем и компонентов, основанные на определении метрик и показателей, связанных с определенными характеристиками качества. В настоящий момент это действующие стандарты, но им на смену готовятся новые стандарты.

Стандарт ISO 12119 (и совпадающий с ним по содержанию IEEE 1465) описывает схему определения требований к различным артефактам жизненного цикла ПО и процесс их проверки с помощью тестирования. Он не действует с 2006 года в связи с пересмотром системы стандартов ISO, нацеленным на гармонизацию стандартов характеристик качества ПО ISO 9126 и процессов их оценки ISO 14598.

Группа стандартов ISO 15504 описывает метод SPICE (Software Process Improvement and Capability dEtermination) оценки и совершенствования процессов разработки программного обеспечения.

Стандарт IEEE 1028 описывает один из методов проведения верификации – *экспертизы* (review). В нем выделены основные виды экспертиз – организационные экспертизы, технические экспертизы, инспекции, сквозной контроль и аудиты, определены роли их участников и возможные методики выполнения. В ходе экспертиз в качестве вспомогательных материалов часто используются списки возможных или наиболее важных видов ошибок в анализируемом ПО.

В целом сложившаяся на данный момент система международных стандартов не содержит целостного подхода к верификации ПО. Большая часть этих стандартов создана на основе опыта разработки сложных программных комплексов для авиакосмической и оборонной отраслей в 70-80-х годах XX века и ориентирована на процессный подход к программной инженерии. Этот подход предполагает сопоставление всех производимых действий с общими

стандартами на разработку ПО, без аккуратного учета специфики предметной области и конкретного проекта, а также детальное планирование и документирование всех операций. В имеющихся стандартах отражены лишь некоторые методы верификации, лучше всего – экспертизы. В частности, достаточно плохо описаны методы тестирования, указывается лишь структура документации и общие процессы подготовки тестов, без систематического изложения содержательных техник разработки тестов и методов оценки полноты тестирования. Кроме того, в имеющихся стандартах совсем не затронуты методы верификации, основанные на привлечении формального анализа свойств ПО, что может быть оправдано достаточно ограниченной пока областью их применения на практике.

Российские стандарты программной инженерии почти всегда являются переводами соответствующих стандартов ISO или IEEE, и поэтому имеют примерно то же содержание.

1.3 Методы верификации программного обеспечения

Методы верификации ПО, в основном нацелены на оценку технических артефактов жизненного цикла. Классификация методов показана на рисунке 3.

1.3.1 Экспертиза

В качестве видов ***общих экспертиз*** (review) выделяют организационные экспертизы (management review), технические экспертизы (technical review), сквозной контроль (walkthrough), инспекции (inspection) и аудиты (audit). С середины 1990-х активно развиваются методы оценки архитектуры ПО на основе сценариев (scenario based software architecture evaluation), обычно не соотносимые с «традиционными» экспертизами. От других методов верификации экспертизу отличает возможность выполнять ее, используя только сами артефакты

жизненного цикла, а не их модели (как в формальных методах) или результаты работы (как в динамических).

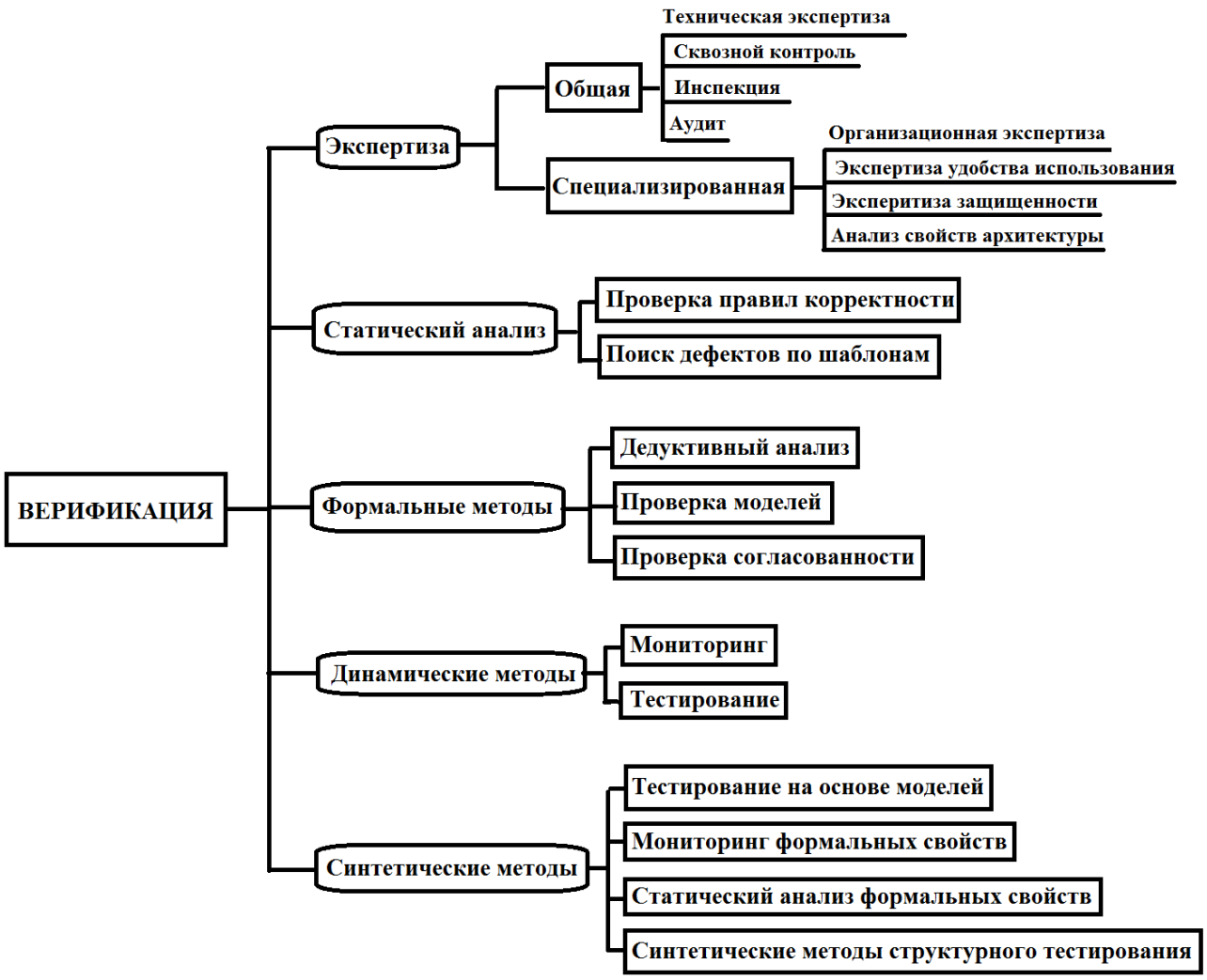

Рисунок 3 – Классификация методов верификации

Экспертиза применима к любым свойствам ПО и любым артефактам жизненного цикла и на любом этапе проекта, хотя для разных целей могут использоваться разные ее виды. Она позволяет выявлять практически любые виды ошибок, причем делать это на этапе подготовки соответствующего артефакта, тем самым минимизируя время существования дефекта и его последствия для качества производных артефактов. В то же время экспертиза не может быть автоматизирована и требует активного участия людей. Эмпирические наблюдения показывают, что эффективность экспертиз в терминах отношения количества обнаруживаемых дефектов к затрачиваемым на это ресурсам несколько

выше, чем для других методов верификации. Так, различные отчеты показывают, что от 50 до 90% всех зафиксированных в жизненном цикле ПО ошибок может быть обнаружено с помощью экспертиз. За счет их раннего обнаружения может быть достигнута существенная экономия ресурсов – затраты на обнаружения ошибки составляют от 5 до 80% от таких же затрат при использовании тестирования. Кроме того, регулярное участие в экспертизах является важным фактором в обучении сотрудников и способствует повышению качества результатов их работы. В то же время эффективность экспертизы существенно зависит от опыта и мотивации ее участников, организации процесса, а также от обеспечения корректного взаимодействия между различными участниками. Это накладывает дополнительные ограничения на распределение ресурсов в проекте и может приводить к конфликтам между разработчиками, если руководство проекта обращает мало внимания на коммуникативные аспекты проведения экспертиз.

Экспертизой ПО (software review, переводится также как критический анализ, рецензирование, просмотр, обзор, оценка и просто анализ) называют все методы верификации, в которых оценка артефактов жизненного цикла ПО выполняется людьми, непосредственно анализирующими эти артефакты.

Традиционно выделяют следующие виды экспертиз.

- *Техническая экспертиза* (technical review) – систематический анализ артефактов проекта квалифицированными специалистами для оценки их внутренней согласованности, точности, полноты, соответствия стандартам и принятым в организации процессам, а также соответствия друг другу и общим задачам проекта.

- *Сквозной контроль* (walkthrough) – метод экспертизы, в рамках которого один из членов команды проверки представляет ее участникам последовательно все характеристики проверяемого артефакта, а они анализируют его, задавая вопросы, внося замечания, отмечая возможные ошибки, нарушения стандартов и другие дефекты.

- *Инспекция* (software inspection) – последовательное изучение характеристик артефакта, обычно следующее некоторому плану, с целью обнаружения в нем ошибок и дефектов.
- *Аудит* (audit) – анализ артефактов и процессов жизненного цикла, выполняемый людьми, не входящими в команду проекта, для оценки соответствия этих артефактов и процессов задачам проекта, заключенному контракту, общим стандартам, друг другу и пр.

Термины «экспертиза» (review) и «инспекция» (inspection) часто употребляются для описания всех перечисленных методов верификации.

Одним из традиционных методов ***специализированных экспертиз*** является организационная экспертиза. *Организационная экспертиза* (management review) – это систематический анализ процессов жизненного цикла и видов деятельности, а также организационных артефактов проекта, выполняемый руководством проекта или от его имени для контроля текущего состояния проекта и оценки выполняемой в его рамках организационной деятельности на соответствие основным целям проекта.

Другим примером специализированного метода является *эвристическая оценка (инспектирование)* пользовательского интерфейса, нацеленная на оценку удобства использования ПО. Она организуется как систематическая оценка различных элементов и аспектов интерфейса с точки зрения определенных эвристик.

В качестве таких эвристик можно использовать определенный набор правил и принципов построения удобных в использовании интерфейсов, или любую достаточно полную систему эвристик, приводимых в руководствах по удобству использования ПО. В рамках одного сеанса оценка интерфейса проводится несколькими специалистами, имеющими опыт в деятельности такого рода. Число оценщиков обычно от трех до пяти человек. Их результаты объединяются в общую картину. В процессе инспектирования разработчики должны отвечать на вопросы, касающиеся различных аспектов как предметной области,

так и работы проверяемой программы. Оценка проводится в два этапа. На первом исследуется архитектура интерфейса в целом, на втором – отдельные контексты взаимодействия и элементы интерфейса. В целом оценка занимает 1-3 часа, после чего проводится анализ полученных результатов, во время которого оценщики описывают обнаруженные ими проблемы и предлагают способы их устранения.

Специализированным видом экспертизы является также *экспертиза или аудит защищенности программных систем*. При проведении экспертиз защищенности используются базы данных известных уязвимостей и дефектов защиты, которые могут быть специфичны для области применения проверяемого ПО и его типа (сетевое приложение, приложение на основе базы данных, компилятор, операционная система и пр.). Важную роль играют также возможные сценарии атак с учетом их вероятностей, стоимости для взломщиков и возможного ущерба для системы.

Особое место среди специализированных методов экспертиз занимают систематические методы *анализа архитектуры* ПО. Первым таким методом, разработанным в 1993 году, стал SAAM (Software Architecture Analysis Method Метод анализа архитектуры ПО). Оценка или сравнение архитектур по этому методу выполняется следующим образом.

1. Определить набор сценариев взаимодействия пользователей или внешних систем с анализируемой. Каждый такой сценарий может использовать возможности, которые уже есть в системе, планируются для реализации или являются новыми. Сценарии должны быть значимы для конкретных заинтересованных лиц – пользователей, разработчиков, представителей заказчика или контролирующей организации, инженеров по сопровождению, и пр. Чем полнее набор сценариев, тем выше будет качество анализа.

2. Определить архитектуру (или несколько сравниваемых архитектур). Это должно быть сделано в форме, понятной всем участникам оценки. Обычно для этого применяются общеупотребительные графические языки, например, UML, однако можно использовать специализированные нотации.

3. Классифицировать сценарии. Для каждого сценария из набора должно быть определено, поддерживается ли он уже данной архитектурой (-ами) или для его поддержки нужно вносить в нее (них) изменения. Сценарий может поддерживаться, т.е. его выполнение не потребует внесения изменений ни в один из компонентов, или же не поддерживаться, если его выполнение требует изменений в описании поведения одного или нескольких компонентов или изменений в их интерфейсах. Поддержка сценария означает, что лицо, заинтересованное в его выполнении, оценивает степень поддержки как достаточную, а необходимые при этом действия – как достаточно удобные.

4. Оценить сценарии. Определить, какие из сценариев полностью поддерживаются рассматриваемыми архитектурами. Для каждого неподдерживаемого сценария надо определить необходимые изменения в архитектуре – внесение новых компонентов, изменения в существующих, изменения связей и способов взаимодействия. Если есть возможность, стоит оценить трудоемкость внесения таких изменений.

5. Выявить взаимодействие сценариев. Определить, какие компоненты требуется изменять для неподдерживаемых сценариев, если требуется изменять один компонент для поддержки нескольких сценариев, эти сценарии называют взаимодействующими. Нужно оценить смысловые связи между взаимодействующими сценариями. Малая связанность по смыслу между взаимодействующими сценариями означает, что компоненты, в которых они взаимодействуют, выполняют слабо связанные между собой задачи и их стоит декомпозировать. Компоненты, в которых взаимодействуют много (> 2-х) сценариев, также являются возможными проблемными местами.

6. Оценить архитектуру в целом (или сравнить несколько заданных архитектур). Для этого надо использовать оценки важности сценариев и степень их поддержки архитектурой. Чем больше сценариев поддерживается, чем меньше трудоемкость изменения архитектуры для поддержки остальных сценариев, тем лучше эта архитектура.

SAAM нацелен, прежде всего, на оценку модифицируемости архитектуры и ее соответствия требованиям, представленным в виде сценариев. Наиболее зрелыми на сегодняшний день являются методы SAAM и ATAM (Architecture Tradeoff Analysis Method Компромиссный метод анализа архитектуры), оба разработаны в Институте программной инженерии (SEI) университета Карнеги-Меллон. Они оба апробированы во многих проектах, в отличие от большинства других предложенных методов. Кроме того, ATAM позволяет оценивать практически любые атрибуты качества ПО за счет привлечения вспомогательных техник на этапе анализа сценариев.

1.3.2 Статический анализ

Статический анализ свойств артефактов жизненного цикла ПО используется для проверки формализованных правил корректного построения этих артефактов и поиска, часто встречающихся дефектов по некоторым шаблонам.

Такой анализ хорошо автоматизируется и может быть практически полностью возложен на инструменты, хотя иногда необходимо вручную определить, например, принятые в проекте стандарты кодирования. Однако применим он лишь к коду или к определенным форматам представления проектных артефактов, и способен обнаруживать только ограниченный набор типов ошибок. Одной из известных проблем статического анализа является также следующая дилемма: либо используются строгие методы анализа, не допускающие пропуска ошибок (тех типов, что ищутся), но приводящие к большому количеству сообщений о возможных ошибках, которые таковыми не являются, либо точным является набор сообщений об ошибках, но возникает возможность пропустить ошибку. Инструменты автоматической верификации на основе статического анализа применяются достаточно широко, поскольку не требуют специальной подготовки и достаточно удобны в использовании. Большинство техник статической проверки корректности программ, доказавших свою эффективность на

практике, рано или поздно становятся частью компиляторов или даже преобразуются в семантические правила языков программирования.

Методы статического анализа артефактов жизненного цикла можно разделить на два вида: контроль того, что все формализованные правила корректности построения этих артефактов выполнены, и поиск типичных ошибок и дефектов в них на основе некоторых шаблонов. Часто инструменты статического анализа используют оба типа проверок. Чаще всего используется статический анализ исходного кода. Проверенные на практике правила корректности кода или шаблоны типичных ошибок переносятся в среды разработки, такие как Eclipse или Microsoft Visual Studio, и постепенно становятся семантическими правилами языков программирования, их проверка возлагается уже на компиляторы этих языков. Поэтому статический анализ можно считать наиболее широко применяемым методом верификации. Если в проекте используются языки описания архитектуры или графические языки проектирования, построенные с их помощью артефакты можно также проверять с помощью специализированных инструментов, например, которые также постепенно встраиваются в широко используемые среды моделирования, такие как Rational Rose. Поэтому методы верификации при помощи статического анализа либо уже прошли апробацию на практике и используются в коммерческих инструментах и широко применяемых инструментах разработки общего назначения, либо все еще остаются в ранге новаторских, исследовательских работ. Исследовательские методы на данный момент, в основном, связаны с формализацией различных характеристик и свойств ПО и поэтому рассматриваются в разделе, посвященным синтетическим подходам к верификации.

1.3.3 Формальные методы верификации

Формальные методы верификации используют для анализа свойств ПО формальные модели требований, поведения ПО и его окружения. Анализ формальных моделей выполняется с помощью специфических техник, таких как

дедуктивный анализ (theorem proving), *проверка моделей* (model checking) или *абстрактная интерпретация* (abstract interpretation). Формальные методы применимы только к тем свойствам, которые выражены формально в рамках некоторой математической модели, а также к тем артефактам, для которых можно построить адекватную формальную модель. Соответственно, для использования таких методов в проекте необходимо затратить значительные усилия на построение формальных моделей. К тому же, построить такие модели и провести их анализ могут только специалисты по формальным методам, которых не так много, и чьи услуги стоят достаточно дорого. Построение формальных моделей нельзя автоматизировать, для этого всегда необходим человек. Анализ их свойств в значительной мере может быть автоматизирован, и сейчас уже есть инструменты, способные анализировать формальные модели промышленного уровня сложности, однако чтобы эффективно пользоваться ими часто тоже требуется очень специфический набор навыков и знаний (в специфических разделах математической логики и алгебры). Тем не менее, в ряде областей, где последствия ошибки в системе могут оказаться чрезвычайно дорогими, формальные методы верификации активно используются. Они способны обнаруживать сложные ошибки, практически не выявляемые с помощью экспертиз или тестирования. Кроме того, формализация требований и проектных решений возможна только при их глубоком понимании, и поэтому вынуждает провести тщательнейший анализ этих артефактов, для чего часто необходима совместная работа специалистов по формальным методам и экспертов в предметной области. В последнее время появились основанные на формальных методах инструменты, решающие достаточно ограниченные задачи верификации ПО из определенного класса, но зато способные эффективно работать в промышленных проектах и требующие для применения минимальных специальных навыков и знаний. Гораздо чаще, чем к программам, формальные методы верификации на практике применяются к аппаратному обеспечению. Их использование в этой области имеет более долгую историю, что привело к созданию более зрелых методик и инструментов. Это обусловлено более высокой стоимостью ошибок

для аппаратного обеспечения, более однородной его структурой и более простыми примитивными элементами, более широким многократным использованием проектной информации, а также большей привычностью строгих ограничений и точных описаний для инженеров.

Формальные методы верификации ПО используют формальные модели требований, поведения и окружения ПО для анализа его свойств. Такие модели являются либо *логико-алгебраическими*, либо *исполнимыми*, либо *промежуточными*, имеющими черты и логико-алгебраических, и исполнимых моделей.

Исполнимые модели (или операционные, executable models) характеризуются тем, что их можно каким-то образом выполнить, чтобы проследить изменение свойств моделируемого ПО. Каждая исполнимая модель является, по сути, программой для некоторой достаточно строго определенной виртуальной машины. Причины моделирования свойств одних программ через свойства других.

Для того, чтобы проверить выполнение тех или иных свойств с помощью формальных методов, необходимо формализовать свойства и проверяемый артефакт, т.е. построить формальные модели для того и другого. Модель проверяемых свойств принято называть *спецификацией*, а модель проверяемого артефакта – *реализацией*. Спецификация и реализация – термины, обозначающие формальные модели, а не описание требований и реализующий их набор программ, как обычно. После этого нужно проверить некоторое формально определенное соответствие или отношение этих моделей, которое моделирует выполнение данных свойств для данного артефакта. Поскольку и реализация, и спецификация могут быть моделями двух основных видов, логико-алгебраическими или исполнимыми, возможно четыре разных комбинации этих видов моделей. Однако на практике никогда не используют для спецификации исполнимую модель в сочетании с логико-алгебраической для реализации, поскольку в такой комбинации невозможно обеспечить большую абстрактность спецификации по сравнению с реализацией. Таким образом, остаются три разных случая.

1. И спецификация S, и реализация I представлены как логико-алгебраические модели. В этом случае выполнение специфицированных свойств в реализации моделируется отношением *выводимости*, что обычно записывается как I ├ S. Чаще всего для его проверки используется метод *дедуктивного анализа* (theorem proving), т.е. проверки того, что набор утверждений, представляющий спецификацию, формально выводится из реализации и, быть может, каких-то гипотез о поведении окружения системы, сформулированных в том же формализме, что и реализация.

2. Спецификация S является логико-алгебраической моделью, а реализация I – исполнимой. Выполнение специфицированных свойств в реализации в этой ситуации называется отношением *выполнимости* и записывается как I ╞ S. Для его проверки используется метод *проверки моделей* (model checking), в рамках которого чаще всего выполнимость проверяется непосредственным исследованием всей реализации, или такой ее части, свойства которой полностью определяют свойства всей реализации в целом. Обычно эту работу выполняет не человек, а специализированный инструмент.

3. И спецификация S, и реализация I представлены как исполнимые модели. В этом случае общепринятого названия или обозначения для выполнения специфицированных свойств в реализации нет – используются термины «симуляция» или «моделирование» (simulation), «сводимость» (reduction), «соответствие» или «согласованность» (conformance).

В тех случаях, когда используются модели промежуточного типа, применяемый метод определяется теми составляющими модели, которые для него наиболее существенны. Так, логики Хоара и динамические логики чаще всего используют для дедуктивного анализа, а программные контракты могут применяться в различных методах.

Дедуктивный анализ используется для верификации логико-алгебраических моделей. Первые методы дедуктивного анализа программ были предложены Флойдом и Хоаром в конце 1960-х годов. В основе этих методов лежит логика Хоара и предложенная Флойдом техника доказательства завер-

шения циклов, основанная на инвариантах цикла и монотонно изменяющихся в ходе его выполнения оценочных функциях. Выполнение верификации программы в методе Флойда организовано следующим образом.

1. Спецификация программы в виде ее предусловия и постусловия определяется формально, например, в рамках исчисления высказываний.

2. В коде программы или на ее блок-схеме выбираются точки сечения, так чтобы любой цикл содержал, по крайней мере, одну такую точку. Начало и конец программы (все возможные точки выхода из программы можно свести к одной) тоже считаются точками сечения.

3. Для каждой точки сечения i находится предикат Φi, характеризующий отношения между переменными программы в этой точке. В начале программы в качестве такого предиката выбирается предусловие, в конце – постусловие. Кроме того, выбирается оценочная функция , отображающая значения переменных программы в некоторое упорядоченное множество без бесконечных убывающих цепей (например, натуральные числа).

4. В результате программа разбивается на набор возможных линейных путей между парами точек сечения. Для каждого такого пути Pij между точками i и j нужно проверить истинность тройки $\{\Phi i\}Pij\{\Phi j\}$. Если это удается, программа частично корректна, т.е. работает правильно, если завершается.

5. Для каждого простого цикла (начинающегося и заканчивающегося в одной и той же точке), нужно найти на нем такой путь Pij, для которого можно доказать $\{\Phi i \ \& \ \varphi = a\}Pij\{\Phi j \ \& \ \varphi < a\}$ для некоторой дополнительной переменной a. Это позволяет утверждать, что цикл завершится, поскольку значения оценочной функции не могут уменьшаться неограниченно.

Достаточно трудной задачей в методе Флойда является нахождение подходящих предикатов. Для ее упрощения Дейкстра предложил использовать технику построения слабейших предусловий, т.е. для программы P и формулы Ψ строить такую формулу $wp(P, \Psi)$, что выполнено $\{wp(P, \Psi)\}P\{ \Psi \}$ и для любой формулы Ψ, если $\{\Phi\}P\{ \Psi \}$, то $\Phi \Rightarrow wp(P, \Psi)$. Для верификации программы при этом достаточно показать, что ее слабейшее предусловие при заданном

постусловии следует из исходного предусловия. В таком виде метод Флойда применим лишь к довольно ограниченному классу программ – в них не должно быть массивов или указателей, вызова подпрограмм, параллелизма, взаимодействия с окружением по ходу работы.

Дедуктивный анализ в принципе может быть выполнен человеком, но для практически значимых систем сам размер спецификации и реализации таков, что необходимо использование специализированных инструментов для автоматического построения доказательств (provers) или предоставляющих существенную помощь в их осуществлении (proof assistants). Поэтому чаще всего в качестве формализма для представления проверяемых свойств и реализации выбирают формализм одного из этих инструментов.

Проверка моделей (model checking) используется для проверки выполнения набора свойств, записанных в виде утверждений какого-либо логико-алгебраического исчисления на исполнимой модели, моделирующей определенные проектные решения или код ПО. Чаще всего для описания проверяемых свойств используется некоторая временная логика или μ-исчисление, а в качестве модели, свойства которой проверяются, выступает конечный автомат, состояния которого соответствуют наборам значений элементарных формул в проверяемых свойствах, обычно он называется *моделью Крипке*. Проверку модели выполняет специализированный инструмент, который либо подтверждает, что модель действительно обладает заданными свойствами, либо выдает сценарий ее работы, в конце которого эти свойства нарушаются, либо не может прийти к определенному вердикту, поскольку анализ модели требует слишком больших ресурсов.

Проверяемые свойства обычно разделяют на *свойства безопасности* (safety properties), означающие, что нечто нежелательное при любом варианте работы системы никогда не случается, и *свойства живучести* (liveness properties), означающие, наоборот, что что-то желательное рано или поздно произойдет. Иногда дополнительно выделяют *свойства стабильности* (или *сохранности*, persistence properties) – при любом сценарии работы системы задан-

ное утверждение в некоторый момент становится истинным и с тех пор остается выполненным – и *свойства справедливости* (fairness properties) – некоторое утверждение при любом сценарии работы будет выполнено в бесконечном множестве моментов времени. Те или иные свойства справедливости, например, что планировщик операционной системы после любого заданного момента времени гарантирует каждому активному процессу получение управления, часто являются исходными предположениями, при выполнении которых нужно проверить свойства безопасности или живучести.

При ***проверке согласованности*** анализируется соответствие между двумя исполнимыми моделями, одна из которых моделирует проверяемый артефакт, обычно проект или реальную работу системы (ее компонента), а вторая – проверяемые свойства. Проверяемые свойства в этом случае – это требования к поведению системы или ее компонента, представленные в виде обобщенного автомата (системы переходов, сети Петри и пр.), все сценарии работы которого объявляются правильными. В этом случае обычно проверяется, что все возможные сценарии поведения реализации возможны. Иногда устанавливается их эквивалентность, т.е. дополнительно проверяется, что все сценарии поведения спецификации есть и у реализации.

Большинство методов и инструментов проверки согласованности используют для этого тестирование, и поэтому относятся к синтетическим методам верификации – к тестированию на основе моделей.

1.3.4 Динамические методы верификации

Динамические методы верификации, в рамках которых анализ и оценка свойств программной системы делаются по результатам ее реальной работы или работы некоторых ее моделей и прототипов. Примерами такого рода методов являются обычное *тестирование* или *имитационное тестирование, мониторинг, профилирование*. Для применения динамических методов необходимо иметь работающую систему или хотя бы некоторые ее компоненты, или же их

прототипы, поэтому нельзя использовать их на первых стадиях разработки. Зато с их помощью можно контролировать характеристики работы системы в ее реальном окружении, которые иногда невозможно аккуратно проанализировать с помощью других подходов. Динамически методы позволяют обнаруживать в ПО только ошибки, проявляющиеся при его работе, а, например, дефекты удобства сопровождения найти не помогут, однако, обнаруживаемые ими ошибки обычно считаются более серьезными. Для применения динамических методов верификации обычно требуется дополнительная подготовка – создание тестов, разработка тестовой системы, позволяющей их выполнять или системы мониторинга, позволяющей контролировать определенные характеристики поведения проверяемой системы. Но системы тестирования, профилирования или мониторинга могут быть сделаны один раз и использоваться многократно для широких классов ПО, лишь сами тесты необходимо готовить заново для каждой проверяемой системы. В то же время подготовка тестов на ранних этапах создания ПО позволяет обнаружить множество дефектов в описании требований и проектных документах – фактически, разработчики тестов вынуждены в ходе своей деятельности выполнять экспертизу артефактов, служащих основой для тестов. Создание набора тестов, позволяющих получить адекватную оценку качества сложной системы, является довольно трудоемкой задачей. Однако среди разработчиков промышленного ПО сложилось (не вполне верное) мнение, что тестирование является наименее ресурсоемким методом верификации, поэтому на практике оно используется для оценки свойств ПО очень широко. При этом чаще всего применяются не слишком надежные, но достаточно дешевые техники, такие как (нестрогое) вероятностное тестирование, при котором тестовые данные генерируются случайным образом, или же тестирование на основе простейших сценариев использования, проверяющие лишь наиболее простые ситуации.

Динамические методы верификации используют результаты реальной работы проверяемой программной системы или ее прототипов, чтобы проверять соответствие этих результатов требованиям и проектным решениям. Существу-

ет два основных вида динамических методов верификации: *мониторинг*, в рамках которого идет только наблюдение, запись и оценка результатов работы ПО при его обычном использовании, и *тестирование*, при котором проверяемое ПО выполняется в рамках заранее подготовленных сценариев. Во втором случае результаты работы тоже записываются, анализируются и оцениваются, основное отличие тестирования от мониторинга – целенаправленные попытки создать определенные ситуации, чтобы проверить работу ПО в них. Как видно, разделение мониторинга и тестирования несколько условно, тестирование всегда включает в себя и мониторинг. Общим для этих методов верификации является создание контролируемой среды выполнения ПО, обеспечивающей получение результатов его работы и измерение различных характеристик ПО, а также оценка этих результатов и характеристик. Динамическая верификация может быть также реальной или имитационной, в зависимости от того, используется в ее ходе само ПО, его прототип или исполнимая модель.

Динамическую верификацию, служащую для обнаружения наличия ошибок и оценки качества ПО, следует отличать от *отладки* (debugging), основная задача которой – определение точного местоположения и исправление ошибок. Однако в ходе разработки динамическая верификация часто используется как часть отладки, и поэтому, помимо самого факта наличия ошибок, должна давать как можно более детальную информацию об их локализации и нарушаемых ими ограничениях. Основное достоинство динамических методов верификации – возможность получить информацию о реальной работе ПО и о реальных показателях его функциональности, производительности, надежности или переносимости. Имитационные методы этого качества лишены и применяются на практике либо для валидации проектных решений, либо для верификации моделей очень сложных систем, для которых эти модели представляют самостоятельную ценность, например, потому, что они впоследствии буду автоматически оттранслированы в исходный код. С помощью динамических методов могут оцениваться все характеристики качества ПО за исключением удобства его сопровождения: функциональность, переносимость, производительность,

надежность и удобство использования. Анализируемые атрибуты качества в первую очередь влияют на построение среды контролируемого выполнения программы для ее тестирования или мониторинга.

При динамической верификации функциональности основное внимание уделяется протоколированию результатов работы операций, доступных элементов состояния компонентов системы, содержимого сообщений, которыми обмениваются компоненты системы, а также действительного порядка событий, насколько это позволяет сделать архитектура системы. При верификации систем реального времени также протоколируются временные интервалы между отдельными событиями.

При контроле переносимости обычно нужно протоколировать такую же информацию, что и при контроле функциональности, или только ту ее часть, которая необходима для проверки одинаковости функционирования системы в разных окружениях.

При анализе производительности протоколируются временные интервалы, в рамках которых система выполняет оцениваемые операции, записываются также объемы памяти (оперативной, в кэшах различных уровней, а также файла подкачки), занимаемые различными компонентами системы при выполнении этих операций, объем и интенсивность обмена данными по сети и пр. Хотя часть этих характеристик может использоваться и при анализе функциональности, в общем случае измерения производительности нужно проводить отдельно, поскольку системы мониторинга функциональности обычно вносят в показатели производительности системы большие искажения. Кроме того, поскольку один вызов часто выполняется за время, сравнимое с точностью измерения времени в компьютерных системах, для корректного определения времени работы отдельных операций часто нужно выполнять много одинаковых вызовов, чтобы получить среднее время их выполнения с небольшой ошибкой, что плохо совмещается с одновременным анализом функциональности.

Для анализа надежности нужны статистические данные по количеству сбоев в системе за определенное время, времени ее доступности для пользова-

телей, среднему времени восстановления при сбоях и пр. С одной стороны, собрать их проще, чем более детальную информацию о функциональности и производительности, но с другой стороны, от механизма ее сбора требуется повышенная надежность и способность корректно протоколировать сбои проверяемой системы.

Динамическая верификация удобства использования практически не применяется, поскольку четко и непротиворечиво сформулировать адекватные требования к удобству использования ПО очень трудно. Чаще всего тестирование и мониторинг удобства использования выполняются для его валидации. Для этого привлекаются пользователи системы или посторонние лица, мало знакомые с системой, но достаточно хорошо знающие предметную область и задачи, для решения которых предназначена проверяемая система. Динамический контроль удобства использования требует специально организованного рабочего места, где было бы удобно протоколировать все действия пользователя, не вмешиваясь в них и никак не отвлекая его от работы с ПО. Обычно для этого используют специальное помещение с полупрозрачным стеклом в одной из стен, чтобы из-за него можно было наблюдать за действиями пользователя. Человек должен быть подготовлен, ему необходимо объяснить, что цель предстоящего мероприятия – оценка удобства системы, а не проверка его способностей и навыков, поэтому все его недоумения и «ошибочные» действия расцениваются как недостатки системы, а не его самого. Все операции пользователя протоколируются (дополнительно к наблюдателям за полупрозрачным стеклом) с помощью видеокамеры, причем удобнее делать это так, чтобы вместе с экраном компьютера было видно лицо пользователя – так ему впоследствии гораздо легче вспомнить и объяснить суть возникавших проблем. Для этого часто используют зеркало, поставленное немного позади экрана компьютера так, чтобы стоящая за плечом пользователя камера снимала бы одновременно экран и выражение лица человека в зеркале. Тестирование удобства использования мало отличается от мониторинга – в первом случае пользователю выдается список задач, которые он должен решить в рамках сеанса тестирования, а при монито-

ринге от него требуется просто разобраться, как работает программа, научиться с ее помощью делать то, что он обычно делает во время работы.

1.3.5 Синтетические методы

Синтетические методы. В последние 15-20 лет появилось множество исследовательских работ и инструментов, в рамках которых применяются элементы нескольких перечисленных выше видов верификации. Так, в отдельные области выделились динамические методы, использующие элементы формальных – *тестирование на основе моделей* (model-based testing, model driven testing) и *мониторинг формальных свойств* (runtime verification, passive testing).

Ряд инструментов построения тестов существенно использует как формализацию некоторых свойств ПО, так и статический анализ кода. Общая идея таких методов вполне понятна – попытаться сочетать преимущества основных подходов к верификации, удалив их недостатки.

Представленная здесь классификация методов верификации скорее обусловлена историческими причинами, чем основана на существенных характеристиках самих этих методов. Исследователи и разработчики новых методов и инструментов, пытаясь соотнести свои работы с работами предшественников, обычно ищут их в рамках одной из указанных групп, поэтому в настоящий момент такая схема достаточно удобна. В последнее время создаются синтетические методы, сочетающие элементы всех остальных, и через 5-10 лет, после появления достаточно большого количества таких подходов, потребуется более детальная и содержательная классификация методов верификации. Далее при обзоре отдельных методов верификации методы, относящиеся к одному типу, рассматриваются вместе. Чаще всего при этом описывается более-менее детально только один представитель этого типа, а также указываются характеристики, по которым методы того же типа могут отличаться друг от друга.

Синтетические методы верификации сочетают техники нескольких типов – статический анализ, формальный анализ свойств ПО, тестирование. Некото-

рые из таких методов породили в последние 10-15 лет самостоятельные бурно развивающиеся области исследований, в первую очередь, *тестирование на основе моделей* и *мониторинг формальных свойств* (runtime verification, passive testing).

Тестирование на основе моделей (model based testing) использует для построения тестов формальные модели требований к ПО и принятых проектных решений. Как критерии полноты тестирования, так и оракулы строятся на основе информации, содержащейся в этих моделях. Получаемые в результате тесты обычно слабо связаны со специфическими особенностями кода тестируемой системы, но содержат представительный набор ситуаций с точки зрения исходной модели. Истоки методологии тестирования на основе моделей лежат в проведенных в 1950-х годах исследованиях возможности определения структуры конечного автомата, моделирующего поведение данной системы, на основе результатов экспериментов с этой системой. Впоследствии были разработаны методы, позволяющие на основе модели в виде конечного автомата строить наборы тестов, успешное выполнение которых при определенных ограничениях на проверяемую систему гарантирует эквивалентность поведения этой системы заданной модели.

В конце 1980-х годов интерес к тестированию на основе моделей возобновляется, и начинают разрабатываться новые подходы, использующие другие виды моделей, помимо конечных автоматов. Основной области их применения в то время было тестирование реализаций телекоммуникационных протоколов на соответствие стандартам этих протоколов. Примерно в это время разрабатывается стандарт ISO 9646 на такое тестирование, учитывающий возможность использования формальных моделей. С середины 1990-х годов тестирование на основе моделей начинает использоваться для других видов программных систем за счет использования более хорошо масштабируемых моделей в виде программных контрактов. В настоящее время методы тестирования на основе моделей используют следующие типы моделей и техник.

Методы проверки согласованности автоматов и систем переходов. Такие методы относятся к одному из трех типов, в зависимости от используемых моделей.

- Обычные и расширенные конечные автоматы. Методы построения тестов на основе конечных автоматов наиболее глубоко разработаны, известны их точные ограничения и гарантии полноты выполняемых проверок. Методы, использующие расширенные автоматы, сводят их к обычным методам, но применяют более детальные критерии покрытия, основанные на использовании данных в расширенных автоматах.
- Системы переходов. Такие методы чаще используются при тестировании распределенных систем, поскольку моделирование таких систем с помощью конечных автоматов очень трудоемко. Большинство этих методов не определяют практически применимых критериев полноты и не дают конечных тестовых наборов для реальных систем, поэтому использующие их инструменты опираются на те или иные эвристики для обеспечения конечности набора тестов.
- Программные контракты. В методах такого рода описание требований к тестируемой системе представляет собой набор программных контрактов, иногда с дополнительным определением явного преобразования состояния системы. Тесты строятся на основе автоматных моделей, являющихся абстракциями от этих контрактов.

Методы построения тестов на основе формального анализа свойств ПО используют формальный анализ для классификации тестовых ситуаций и нацеленной генерации тестов.

- Методы на основе проверки моделей. В рамках таких методов тестовые ситуации выбираются как представители классов эквивалентности, задаваемых критерием покрытия. Соответствующая ситуации тестовая последовательность строится как контрпример при проверке модели (model checking) на

выполнение свойства, являющегося отрицанием условия достижения этой ситуации.

- Методы на основе дедуктивного анализа. В этих методах выбираемые тестовые ситуации соответствуют особым случаям в дедуктивном анализе свойств тестируемой системы.

Методы построения тестов с помощью символического выполнения (symbolic execution). Такие методы используют символическое описание проходимого во время выполнения теста пути по коду программы (или формальных проверяемых спецификаций) в виде набора предикатов. Это описание позволяет выбирать новые тестовые ситуации так, чтобы они покрывали другие пути и строить тесты с помощью техник *разрешения ограничений* (constraint solving). Символическое выполнение в комбинации с конкретизацией тестовых данных используется и для построения тестов, нацеленных на типичные дефекты, такие, как использование неинициализированных объектов, тупики и гонки параллельных потоков.

Более детальную таксономию методов и инструментов тестирования на основе моделей. Можно отметить, что наиболее развиты методы тестирования на основе моделей первой из перечисленных выше групп. Поддерживающие их инструменты все шире начинают использоваться в промышленных проектах. Методы построения тестов на основе проверки моделей и на основе символического выполнения активно развиваются и начинают воплощаться в инструменты.

Мониторинг формальных свойств ПО (для этой области используется два английских термина – runtime verification и passive testing) стал развиваться как отдельное направление исследований в конце 1990-х годов. В основе этого подхода лежит фиксация формальных свойств ПО, которые необходимо проверить и встраивание их проверок в систему мониторинга. В дальнейшем выполняется мониторинг ПО, в ходе которого контролируются и выделенные свойства.

Использование описание свойств с помощью обычных и временных логик. Подобные техники осуществляют контроль свойств, записанных в виде формул временных логик, с помощью записи событий, происходящих в работающем ПО и анализа возникающих трасс.

Использующие описание свойств в виде систем переходов или автоматов. В этих техниках трасса анализируется не на выполнение некоторых формул, а на согласованность с заданной автоматной моделью правильного поведения.

Использующие программные контракты. В рамках таких подходов корректность поведения системы проверяется во время ее работы с помощью оракулов на базе программных контрактов, внедренных в систему мониторинга.

Пока методы мониторинга формальных свойств используются не в исследовательских целях достаточно редко, в основном, для мониторинга небольших критических приложений. Так, NASA применяет несколько таких инструментов, в том числе, разработанный силами собственных сотрудников Java PathExplorer, для верификации систем управления спутниковыми системами.

Методы статического анализа формальных свойств исследуются уже несколько десятилетий, но только в последние 7-10 лет появились реализующие их инструменты, пригодные для применения к сложным программным системам. По используемым этими инструментами техникам формального анализа их можно разделить на следующие группы.

Методы на основе генерации верификационных утверждений (verification conditions) и их дедуктивного анализа. Подобные методы статического анализа были названы *расширенным статическим анализом* (extended static checking). На них основаны такие инструменты как ESC/Java 2 или Boogie, требующие пополнения кода ПО предусловиями и постусловиями отдельных функций (часто также инвариантами типов данных и циклов), и не требующие вмешательства человека инструменты, такие как Saturn или Calysto, нацеленные на поиск типичных ошибок.

Методы на основе проверки свойств моделей, автоматически создаваемых на основе исходного кода. Такие методы используют так называемую *абстрактную интерпретацию* (abstract interpretation) для построения простых моделей поведения кода, которые позволяют проверить определенные свойства, но могут терять информацию, касающуюся других свойств. Такие методы, в свою очередь, делятся на две группы.

- Методы на основе булевских моделей. Эти модели являются конечными наборами булевских значений, выбираемыми так, чтобы достаточно точно можно было анализировать возможные сценарии выполнения программы. Такие модели используются в инструменте Blast и в SLAM, который был переработан в Microsoft в Static Driver Verifier, используемый для эффективного поиска ошибок, связанных с использованием ресурсов операционной системы в драйверах Microsoft Windows.
- Методы на основе разнородных моделей. Такие техники используют более сложные модели, чем булевские (например, представление областей возможных значений переменных в виде выпуклых многогранников или деревья выбора значений числовых переменных в зависимости от набора булевских), которые позволяют эффективнее анализировать специфические свойства ПО. К инструментам такого рода относятся ASTREE и PolySpace Verifier.

Хотя инструменты статического анализа формальных свойств появились не так давно, некоторые из них (SLAM в виде Static Driver Verifier) уже используются в промышленном программировании. При практическом использовании основным недостатком инструментов статического анализа становится большое количество сообщений о возможных ошибках или дефектах, которые таковыми не являются. При работе с программными системами размеров в несколько сотен тысяч строк могут возникать тысячи таких сообщений, проанализировать их вручную в этих случаях не представляется возможным. Поэтому большое количество исследований ведется в направлении создания инструментов, вы-

дающих как можно меньше ложных сообщений об ошибках, и в то же время, не пропускающих серьезные дефекты.

Синтетические методы генерации структурных тестов

В последнее время активно разрабатываются инструменты автоматической генерации тестов на основе кода, которые используют дополнительные источники информации. В качестве таких источников выступают статический анализ кода, формальный анализ, мониторинг выполнения ранее построенных тестов и т.п. Поскольку в инструментах этого типа используется обычно 3-4 техники разных типов по используемой в этом обзоре классификации, методы, лежащие в их основе, вынесены в отдельную разновидность синтетических методов верификации. Характерным для этих инструментов примером является развитие инструмента JCrasher, созданного в 2003-2004 годах в университете Орегона, впоследствии переработанного сначала в Check-n-Crash, а затем – в DSDCrasher.

JCrasher генерирует структурные тесты для Java-программ, используя случайные данные примитивных типов, несколько простых эвристик нацеливания на вероятные ошибки, синтаксис операций и структуру данных. Получаемый тест представляет собой последовательность вызовов операций, аргументы которых могут быть сгенерированы случайно или получены как результаты предыдущих вызовов. Выполняемые проверки сводятся к отсутствию исключений и сбоев.

В инструменте Check-n-Crash добавлен предварительный этап статического анализа реализации тестируемых операций, на котором выделяются предикаты, соответствующие различным путям выполнения операций, которые затем разрешаются частично с помощью разрешения ограничений, частично за счет небольших модификаций случайных тестов.

DSDCrasher добавляет еще одну предварительную фазу, на которой тестируемая программа выполняется на множестве случайных сценариев, и с помощью дополнительного инструмента Daikon выявляются ее возможные инварианты и ограничения. Затем они используются для отсеивания некорректных

сценариев тестирования, которые приводят к ошибкам не в силу ошибочной работы программы, а из-за неправильного ее использования.

Другие примеры подобной интеграции различных техник верификации в инструментах генерации структурных тестов дают инструменты Randoop и SMART. Они также основаны на случайной генерации последовательностей тестовых вызовов, нацеливаемой на сложные ошибки взаимодействия. В Randoop это нацеливание происходит за счет сокращения множества возможных состояний тестируемой программы при помощи символического анализа выполнения получаемых тестов и использования эвристик попадания в новые ситуации – иногда генерируются длинные последовательности одинаковых вызовов. SMART вместе с символическим выполнением использует выделение возможных ограничений, как и DSDCrasher.

2 Практическая часть дисциплины

Цель практических и лабораторных работ: закрепление теоретических знаний и приобретение практических навыков.

2.1 Задания для практических занятий

Практическая работа №1. Экспертиза технического задания ПС

Согласно варианту провести экспертизу по соответствию, технического задания и разработанного ПО (по реализованным функциям и критериям качества).

Практическая работа №2. Экспертиза проектной части ПС

Согласно варианту провести экспертизу по соответствию проектной части работы с реализованным ПО.

Практическая работа №3. Проверка артефактов ПС

Согласно варианту проверить соответствие создаваемых в ходе разработки ПО артефактов стандартам, описанием требований (техническое задание) к ПО, проектным решениям, исходным кодам, пользовательской документации и функционированию самого ПО (верификация).

Практическая работа №4. Валидация ПС

Согласно варианту проверить соответствие создаваемых и используемых в ходе разработки ПО артефактов нуждам и потребностям пользователей и заказчиков с учетом законов предметной области и ограничений контекста использования ПО.

Практическая работа №5. Общая экспертиза

Согласно варианту в зависимости от предметной области и решаемых задач выбрать наиболее приемлемые виды общих экспертиз (технические, сквозной контроль, инспекции, аудиты), а также специализированных (организационная, удобства использования, защищенности, анализ архитектуры).

Описать какие инструментальные средства применялись для статического анализа

2.2 Задания для лабораторных занятий

Лабораторная работа №1. Динамические методы верификации

Согласно варианту описать использование динамических методов (мониторинг, тестирование)

Лабораторная работа №2. Функциональное тестирование ПС

Согласно варианту провести тестирование ПС по методу «черного ящика» (функциональное, объемно-нагрузочное, классы эквивалентности, граничные значения, причинно-следственные связи, состояние переходов, отказ и восстановление, юзабилити-тестирование)

Лабораторная работа №3. Структурное тестирование ПС

Согласно варианту провести тестирование по методу «белого ящика» (базового пути, условий, ветвей и операторов отношений, потоков данных, циклов, уровни тестового покрытия).

2.3 Курсовая работа

Верификация и валидация ПО

Согласно варианту, выполнить следующее:

- разработать техническое задание на ПС;
- выполнить проектную часть ПС: разработать архитектуру, структуру данных, алгоритм решения поставленной задачи, функциональную реализацию;
- реализовать ПС;
- провести верификации и валидацию, разработанного ПС.

Верификацию и валидацию провести в следующем порядке.

1 Провести экспертизу по соответствию, технического задания и разработанного ПО (по реализованным функциям и критериям качества).

2 Провести экспертизу по соответствию проектной части работы с реализованным ПО.

3 Проверить соответствие создаваемых в ходе разработки ПО артефактов стандартам, описанием требований (техническое задание) к ПО, проектным решениям, исходным кодам, пользовательской документации и функционированию самого ПО (верификация).

4 Проверить соответствие создаваемых и используемых в ходе разработки ПО артефактов нуждам и потребностям пользователей и заказчиков с учетом законов предметной области и ограничений контекста использования ПО (валидация).

5 В зависимости от предметной области и решаемых задач выбрать наиболее приемлемые виды общих экспертиз (технические, сквозной контроль, инспекции, аудиты), а также специализированных (организационная, удобства использования, защищенности, анализ архитектуры).

6 Описать какие инструментальные средства применялись для статического анализа.

7 Описать использование динамических методов (мониторинг, тестирование).

Заключение

В методических указаниях дан обзор современных методов верификации программного обеспечения. В ней обсуждается место верификации в проектах по разработке и сопровождению ПО, а также различные способы ее выполнения, как широко применяемые в промышленной практике, так и используемые пока только в исследовательских проектах. В круг рассматриваемых методов входят различные виды экспертиз, формальные методы верификации, тестирование и мониторинг, статический анализ. Для каждого разбираемого метода приводится информация о поддерживающих его инструментах.

Приведены задания для практических и лабораторных работ с учетом теоретического материала. Приведены все этапы выполнения курсовой работы.

Список использованных источников

1 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 – P. 81-90.

2 Демин, А. А. Методы верификации и валидации сложных программных систем / А.А. Демин, А.А. Карпунин, Ю.М. Ганев //Программные продукты и системы, 2014, - №4, - с. 229-233.

3 Зубкова, Т. М. Технология разработки программного обеспечения: [Электронный ресурс]: учебное пособие для обучающихся по образовательным программам высшего образования по направлениям подготовки 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия, 09.03.02 Информационные системы и технологии / Т. М. Зубкова; М-во образования и науки Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования "Оренбург. гос. ун-т", Каф. прогр. обеспечения вычисл. техники и автоматизир. систем. - Электрон. текстовые дан. (1 файл: 3.71 Мб). - Оренбург : ОГУ, 2017. - 468 с. режим доступа: http://artlib.osu.ru/site_new/trudi

4 Корнейчук, О.С. Методы верификации программного обеспечения/О.С. Корнейчук// Прикладная математика и фундаментальная информатика, 2006, - №3, - с.163-167.

5 Кузнецов, А. С. Многоэтапный анализ архитектурной надежности и синтез отказоустойчивого программного обеспечения сложных систем [Электронный ресурс]: монография / А. С. Кузнецов, С. В. Ченцов, Р. Ю. Царев. - Красноярск: Сиб. федер. ун-т, 2013. - 143 с. - ISBN 978-5-7638-2730-9. Режим доступа: http://znanium.com/bookread.php?book=492347

6 Кулямин, В.В. Методы верификации программного обеспечения/В.В. Кулямин //Всероссийский конкурс обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы», 2008. - 117 с. Режим доступа: http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf

7 Орлов, С. А. Технологии разработки программного обеспечения: разработка сложных программных систем: учеб. для вузов / С. А. Орлов .- 3-е изд. - СПб. [и др.] : Питер, 2004. - 527 с.: ил. - ISBN 5-94723-145-X

8 Основы теории надежности информационных систем: учебное пособие / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2013. - 256 с. - (Высшее образование). ISBN 978-5-8199-0563-0. - Режим доступа: http://znanium.com/bookread.php?book=419574

9 Оценка качества программного обеспечения: Практикум: Учебное пособие / Б.В. Черников, Б.Е. Поклонов; Под ред. Б.В. Черникова - М.: ИД ФОРУМ: НИЦ Инфра-М, 2012. - 400 с. - (Высшее образование). (п) ISBN 978-5-8199-0516-6. - Режим доступа: http://znanium.com/bookread.php?book=315269

10 Плаксин, М.А. Тестирование и отладка программ для профессионалов будущих и настоящих [Электронный ресурс] : . - Электрон. дан. - М. : Лаборатория знаний, 2013. - 168 с. - Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=42625 ? Загл. с экрана. http://e.lanbook.com/view/book/42625/

11 Технология программирования : учебник / Г.С. Иванова. - Москва : КноРус, 2011. - 333 с. - ISBN 978-5-406-00519-4.

12 Технология разработки программного обеспечения: Учеб. пос. / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул; Под ред. проф. Л.Г.Гагариной - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013. - 400 с.: ил.; 60x90 1/16. - (Высшее обр.). (п) - ISBN 978-5-8199-0342-1. - Режим доступа: http://znanium.com/catalog_/product/389963

13 Управление качеством программного обеспечения: Учебник / Б.В. Черников. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 240 с. - (Высшее образование) - ISBN 978-5-8199-0499-2. - Режим доступа: http://znanium.com/go.php?id=256901