КАФЕДРА ИВС

Тема: «Тестирование программного кода. Задачи и цели тестирования программного кода »

Дисциплина: «Верификация, стандартизация и сертификация ПО»

Лектор: старший преподаватель кафедры ИВС Савченко Н.К.

План лекции:

1. Методы верификации программного обеспечения.
2. Тестирование.

2.1 Виды тестирования.

3. Критерии полноты тестирования.

1 Методы верификации программного обеспечения

Методы верификации ПО, в основном нацелены на оценку технических артефактов жизненного цикла.

Такие методы разделяются на следующие группы (рисунок 2.1):

- Экспертиза;
- Статистический анализ;
- Формальные методы;
- Динамические методы;
- Статистические методы.

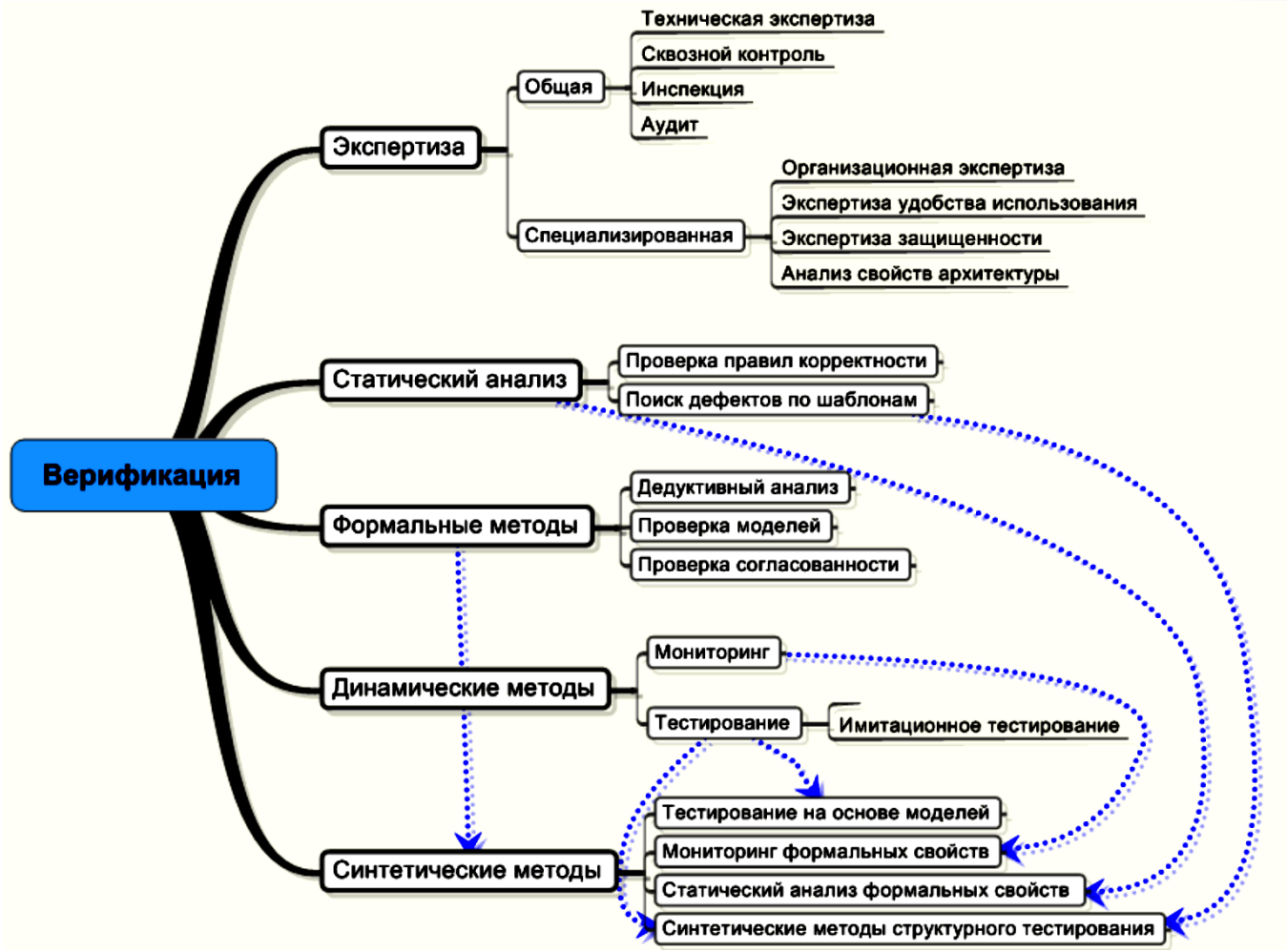

Рисунок 2.1 - Методы верификации ПО

Экспертиза (review) различных артефактов жизненного цикла ПО. Обычно в качестве видов экспертиз выделяют организационные экспертизы (management review), технические экспертизы (technical review), сквозной контроль (walkthrough), инспекции (inspection) и аудиты (audit).

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).

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.

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.

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. 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.

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.

Синтетические методы. В последнее время 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, купировав их недостатки.

2 Тестирование

Тестирование (testing) является методом верификации, в рамках которого результаты работы тестируемой системы или компонента в ситуациях из выделенного конечного набора проверяются на соответствие проектным решениям, требованиям, общим задачам проекта, в рамках которого эта система разрабатывается или сопровождается.

Ситуации, в которых выполняется тестирование, называют *тестовыми ситуациями* (test situations, test purposes), а процедуры, описывающие процесс создания этих ситуаций и проверки, которые необходимо выполнить над полученными результатами, — *тестами*.

Тестирование может использоваться и для валидации, в этом случае оно проверяет соответствие поведения ПО ожиданиям и потребностям пользователей и заказчиков, а для оценки этого соответствия должны привлекаться сами пользователи, их представители, бизнес-аналитики или эксперты в предметной области.

Тестирование, как и верификация вообще, служит для поиска ошибок или дефектов и для оценки качества ПО. Эффективность решения обеих этих задач во многом определяется тем, какой именно набор тестовых ситуаций выбран для проведения тестирования.

Чтобы иметь некоторые гарантии аккуратности полученных в ходе тестирования оценок качества, необходимо выбирать тестовые ситуации систематическим образом, в соответствии с основными задачами и рисками проекта. Правила, определяющие набор необходимых тестовых ситуаций, называют *критериями полноты (*или *адекватности) тестирования* (test adequacy criteria).

Обычно такой критерий использует разбиение всех возможных при работе проверяемого ПО ситуаций на некоторые классы эквивалентности, такие, что ситуации из одного класса достаточно похожи друг на друга и работа ПО в них не должна отличаться сколь-нибудь значительным образом. Полноту тестирования при этом можно определять по проценту задействованных, «покрытых» в нем классов ситуаций.

Такой критерий полноты называется *критерием тестового покрытия* (test coverage criterion), а процент покрытых в результате работы тестов классов ситуаций — *достигнутым тестовым покрытием*.

Различных критериев полноты довольно много, а тестирование, покрывшее 100% выделенных по одному из таких критериев классов ситуаций, и поэтому часто называемое *полным*, в действительности не является полным или исчерпывающим в каком-либо смысле, связанным со строгими гарантиями отсутствия ошибок и соответствия ПО требованиям. Такого рода исчерпывающее тестирование невозможно для практически значимых систем.

Подготовка и проведение тестирования в проекте создания или сопровождения ПО проходят примерно по следующему плану.

1. Определение целей тестирования (test objectives, test goals) на основе задач и рисков проекта. Такие цели очерчивают проверяемые характеристики и свойства ПО, тщательность тестирования отдельных компонентов и подсистем. Они также определяют используемые в проекте виды тестирования и методы построения тестов.

2. Определение требований, свойств и ограничений, которые должны проверяться в ходе тестирования.

3. Определение критерия полноты тестирования, который будет использоваться в данном проекте. Критерий полноты должен быть согласован с целями тестирования, он управляет выбором тестовых ситуаций для тестирования, а также определяет, когда можно прекратить тестирование.

3. Построение набора тестов, нацеленных на достижение выбранного критерия полноты и проверяющих определенные ранее требования и ограничения.

4. Отладка, выполнение тестов и получение итоговых результатов тестирования в виде сообщений о выполненных действиях и нарушениях проверяемых ограничений. Отладка тестов включает пробные прогоны, устранение обнаруживаемых ошибок в самих тестах, а также описаниях проектных решений и требований, на основе которых получены тесты. По результатам отладки может потребоваться пополнение набора тестов для достижения максимального возможной полноты тестирования в рамках выделенных ресурсов.

5. Анализ результатов тестирования, составление отчетов о найденных дефектах и о полученных оценках качества ПО.

2.1 Виды тестирования

Классификация видов тестирования достаточно сложна, потому что может проводиться по нескольким разным аспектам.

По уровню или масштабу проверяемых элементов системы тестирование делится на следующие виды.

Модульное или *компонентное* (unit testing, component testing) — проверка корректности работы отдельных компонентов системы, выполнения ими своих функций и предполагаемых проектом характеристик.

Интеграционное (integration testing) — проверка корректности взаимодействий внутри отдельных групп компонентов.

Системное (system testing) — проверка работы системы в целом, выполнения ею своих основных функций, с использованием определенных ресурсов, в окружении с заданными характеристиками.

По проверяемым характеристикам качества тестирование может быть тестирование функциональности, производительности (и по времени, и по другим ресурсам), надежности, переносимости или удобства использования.

Более специфические виды тестирования, нацеленные на оценку отдельных атрибутов, — тестирование защищенности, совместимости или восстановления при сбоях.

Специфическим видом тестирования, нацеленным на минимизацию риска того, что в результате доработки или внесения ошибок качество системы изменилось в худшую сторону, является *регрессионное тестирование* (regression testing). При его проведении используется уже применявшийся ранее набор тестов, и оно должно выявить различия между результатами, полученными на этих тестах ранее, и наблюдаемыми после внесения изменений.

По источникам данных, используемых для построения тестов, тестирование относится к одному из следующих видов.

Тестирование черного ящика (black-box testing, часто также называется *тестированием соответствия*, conformance testing, или *функциональным тестированием* (functional testing) — нацелено на проверку соблюдения требований.

Использует критерии полноты, основанные на требованиях, и техники построения тестов, использующие только информацию, заданную в требованиях к проверяемой системе.

Частными случаями этого вида тестирования являются тестирование на соответствие стандартам и *квалификационное* или *сертификационное тестирование*, нацеленное на получение некоторого сертификата соответствия определенным требованиям или стандартам.

Тестирование белого ящика (white-box testing, glass-box testing, также *структурное тестирование*, structural testing) — нацелено на проверку корректности работы кода. Использует критерии полноты и техники построения тестов, основанные на структуре проверяемой системы, ее исходного кода.

Тестирование серого ящика (grey-box testing) использует для построения тестов как информацию о требованиях, так и коде. На практике оно встречается чаще, чем предыдущие крайние случаи.

Тестирование, нацеленное на ошибки — использует для построения тестов гипотезы о возможных или типичных ошибках в ПО такого же типа, как проверяемое.

К этому типу относятся, например, следующие виды тестирования.

- Тестирование работоспособности (sanity testing, smoke testing), нацеленное на проверку того, в систему включены все ее компоненты и операции, и система не дает сбоев при выполнении своих основных функций в простейших сценариях использования.
- Тестирование на отказ, пытающееся найти ошибки в ПО, связанные с контролем корректности входных данных.

- Нагрузочное тестирование (load testing), проверяющее работоспособность ПО при больших нагрузках — больших объемах входных, выходных или промежуточных данных, большой сложности решаемых задач, большом количестве пользователей, работающих с ПО и пр.
- Тестирование в предельных режимах (stress testing), проверяющее работоспособность ПО на границах его возможностей и на границах той области, где оно должно использоваться.

По роли команды, выполняющей тестирование, оно может относиться к следующим видам.

- *Внутреннее тестирование* выполняется в рамках проекта по разработке системы силами организации-разработчика ПО.
- *Независимое тестирование* выполняется третьими лицами (не разработчиками, не заказчиками и не пользователями) для получения объективных и аккуратных оценок качества системы.
- *Аттестационное тестирование* (приемочные испытания) выполняется представителями заказчика непосредственно перед приемкой системы в эксплуатацию для проверки того, что основные функции системы реализованы. Обычно аттестационные тесты являются очень простыми тестами функциональности и производительности.

Пользовательское тестирование осуществляется силами пользователей системы. У него есть два часто упоминаемых частных случая.

- Альфа-тестирование (alpha-testing) выполняется самими разработчиками, но в среде, максимально приближенной к рабочему окружению системы и на наиболее вероятных сценариях ее реального использования.
- Бета-тестирование (beta-testing) выполняется пользователями, желающими познакомиться с возможностями системы до ее официального выпуска и передачи в эксплуатацию.

3 Критерии полноты тестирования

Критерии полноты тестирования должны выбираться на основе рисков проекта и того, какие ошибки в тестируемом ПО наиболее важны. На практике используют критерии следующих типов.

Критерии покрытия структурных элементов тестируемой системы, которые выполняются или задействуются в ходе тестов. Такие критерии называются *структурными критериями* и используются, если код тестируемой системы доступен для анализа.

Структурные критерии покрытия позволяют отслеживать, были ли отдельные элементы кода хотя бы раз выполнены в ходе тестирования, однако не могут ничем помочь в обнаружении нереализованных требований.

На уровне отдельных операций используют *критерии покрытия кода*, основанные на элементах графа потока управления или графа потоков данных.

К первому типу относятся *критерий покрытия инструкций или ветвлений в коде*, критерий покрытия комбинаций элементарных условий, используемых в ветвлениях.

Критерий покрытия использований переменных или критерий покрытия путей от точек присваивания переменной некоторого значения к точкам ее использования относятся к критериям покрытия потоков данных.

На уровне компонента или группы компонентов используют покрытие элементов графа вызовов их операций или вхождений операций чтения и записи отдельных элементов данных этих компонентов в код тестируемых операций.

На уровне системы в целом или крупных подсистем используют критерии покрытия возможных сценариев взаимодействия компонентов.

Специальным частным случаем структурных критериев являются *критерии покрытия, основанные на структуре входных данных тестируемых операций*.

Эти критерии полезны в тех случаях, когда код системы для анализа недоступен, или описание требований дает недостаточно информации для построения аккуратных тестов.

Критерии покрытия элементов требований. Они называются *функциональными критериями* и устроены подобно структурным критериям покрытия, только в качестве покрываемых элементов рассматриваются не сценарии взаимодействия компонентов, участки их кода или комбинации условий ветвлений, а сценарии использования, отдельные правила и ограничения в требованиях и комбинации условий, при которых задействуются эти правила и ограничения.

Такие критерии полноты позволяют отслеживать, что тестирование проверяет все ограничения, указанные в требованиях, однако они не позволяют целенаправленно обнаружить дополнительные функции или закладки в коде, что можно сделать, используя структурные критерии.

Контрольные вопросы для СРС:

1. **Охарактеризуйте основные методы верификации программного обеспечения.** В чем заключается отличие тестирования от статических методов верификации (инспекции, ревью, статического анализа кода)?
2. **Сравните основные виды тестирования программного обеспечения.** Дайте характеристику модульному, интеграционному, системному и приемочному тестированию, а также тестированию методом «черного» и «белого» ящика.
3. **Что понимается под критериями полноты тестирования?** Рассмотрите критерии покрытия операторов, ветвей, условий и путей. Объясните, в каких случаях каждый из них используется и какой обеспечивает наиболее полную проверку программного кода.

Список рекомендуемой литературы

1. В.В. Кулямин. Методы верификации программного обеспечения. - М.: Институт системного программирования РАН, 2019. – 211 с.

2. В.В. Липаев. Программная инженерия. – М.: Гос. ун-т - Высшая школа экономики. – ТЕИС, 2016. – 608 с.

3. С.В. Синицын. Верификация программного обеспечения: учебное пособие. – Саратов: Профобразование, 2019. — 368 с.

Лекция окончена.

Спасибо за внимание!