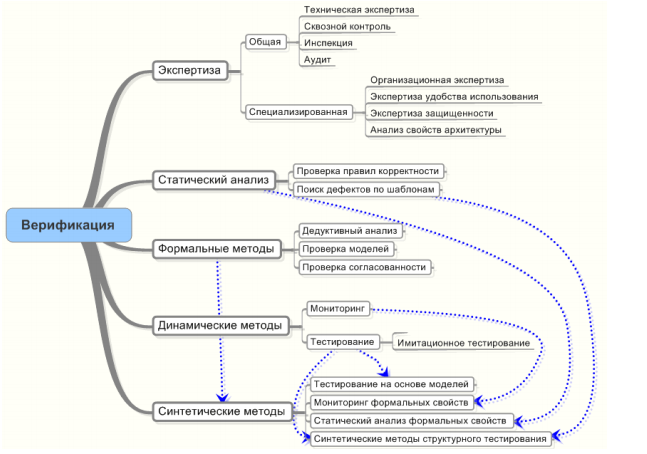
1. 4. Классификация методов верификации ПО. Место и роль статического анализа.

В данном разделе рассматриваются методы верификации ПО, в основном нацеленные на оценку технических артефактов жизненного цикла. Такие методы в имеющейся литературе разделяются на следующие группы.



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). 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). 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 [56-61]. 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 [59- 63]. 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 [58-61], 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. 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. • 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 последние 10 лет появились основанные на формальных методах инструменты [64-68], 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 [69-72]. 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. • 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 последние 10-15 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) [73] 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. 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. Однако, как уже было сказано, 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.

В большинстве случаев под статическим анализом подразумевают анализ, осуществляемый с помощью автоматизированных инструментов исходного или исполняемого кода.

Исторически первые инструменты статического анализа (часто в их названии используется слово lint) применялись для нахождения простейших дефектов программы. Они использовали простой поиск по сигнатурам, то есть обнаруживали совпадения с имеющимися сигнатурами в базе проверок. Они применяются до сих пор и позволяют определять "подозрительные" конструкции в коде, которые могут вызвать падение программы при выполнении.

Недостатков у такого метода немало. Основным является то, что множество "подозрительных" конструкций в коде не всегда являются дефектами. В большинстве случаев такой код может быть синтаксически правильным и работать корректно. Соотношение "шума" к реальным дефектам может достигать 100:1 на больших проектах. Таким образом, разработчику приходится тратить много времени на его отсеивание от реальных дефектов, что отменяет плюсы автоматизированного поиска.

Несмотря на очевидные недостатки, такие простые утилиты для поиска уязвимостей до сих пор используются. Обычно они распространяются бесплатно, так как коммерческого применения они, по понятным причинам, не получили.

Второе поколение инструментов статического анализа в дополнение к простому поиску совпадений по шаблонам оснащено технологиями анализа, которые до этого применялись в компиляторах для оптимизации программ. Эти методы позволяли по анализу исходного кода составлять графы потока управления и потока данных, которые представляют собой модель выполнения программы и модель зависимостей одних переменных от других. Имея данные, графы можно моделировать, определяя, как будет выполняться программа (по какому пути и с какими данными).

Поскольку программа состоит из множества функций, процедур модулей, которые могут зависеть друг от друга, недостаточно анализировать каждый файл по отдельности. Для полноценного межпроцедурного анализа необходимы все файлы программы и зависимости.

Основным достоинством этого типа анализаторов является меньше количество "шума" за счет частичного моделирования выполнения программ и возможность обнаружения более сложных дефектов.

1. Цели, задачи, возможности, ограничения статического анализа исходных кодов ПО.