Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 10. С. 235–251.

DOI: **10.7463/1015.0823129**

Представлена в редакцию: 28.04.2015
Исправлена: 05.09.2015

УДК 004.3+519.6

Методы верификации программного обеспечения

Гурин Р. Е.[1,*], **Рудаков И. В.**[1], **Ребриков А. В.**[1]

[*]rg.bmstu@gmail.com

[1]МГТУ им. Н.Э. Баумана, Москва, Россия

Работа посвящена обзору и классификации методов верификации программного обеспечения (ПО). Проведен обзор существующих статических методов верификации, рассмотрены характеристики методов и реализация анализа выявления зависимостей в рамках метода абстрактной интерпретации. В статье представлен обзор и классификация динамических методов верификации ПО, включающий в себя обзор видов инструментирования кода программы. Классификация проведена по ряду важнейших признаков, таких как: вид метода, уровень автоматизации, уровень функциональной пригодности, точность, типы обнаруживаемых ошибок, эффективность, область применимости метода, время выполнения и способ достижения результата. В статье приведено описание методов и их реализации. Проведен обзор видов тестирования и средств логического вывода и рассмотрены основные особенности методов верификации. Выявлены недостатки существующих методов верификации программ.

Ключевые слова: верификация, анализ кода, статический анализ, динамический анализ, интерпретация, символьное выполнение, проверка модели

Введение

Одной из важнейших проблем при проектировании и разработке программного обеспечения (ПО) является его верификация. Методы верификации программного обеспечения предназначены для подтверждения фактов соответствия конечного программного продукта заявленным требованиям, целью верификации программного обеспечения является обнаружение ошибок, уязвимостей, некорректно реализованных свойств и требований [1]. Проблема создания новой классификации методов верификации ПО является актуальной, так как позволяет рассмотреть существующие на данные момент методы верификации ПО и их программную реализацию, выявить их преимущества и недостатки. Классификация и исследование существующих методовпозволяет составить список требований и рекомендаций для дальнейшей разработки и исследования синтетического метода верификации ПО, на основе SMT – решателя.

Существующие методы верификации[2] ПО можно разделить на эмпирические (использующие экспертизу), формальные (использующие математический аппарат для

верификации программного обеспечения) и динамические (проверяющие работу программной реализации с помощью непосредственного запуска), а с точки зрения уровня автоматизации на ручные, автоматизированные и автоматические.

1. Верификация программного обеспечения

Одним из предназначений верификации ПО является проверка соответствия реализованного программного кода техническому заданию и требования к функциональности [2].

Экспертиза используется для проверки документации и кода программного обеспечения на соответствие с нормами и стандартам оформлениями, которые приняты в стране, отрасли и организации. Экспертиза может быть общей и специализированной.

Термин верификации ПО в одной из нотаций [3] означает*символьное выполнение программы* или проверку кода на наличие ошибок и уязвимостей методами проверки модели.

Методы формальной верификации позволяют провести верификацию программного обеспечения основываясь на математической модели программы без обращения к ее физической реализации.

Техника символьного выполнения [4-5]широко используется в настоящее время для тестирования и анализа программ [5,6] и позволяет проводить моделирование выполнения программы, при котором часть входных переменных представляется в символьном виде. Символ переменной обозначает множество значений входной переменной программы из области ее определения. Каждое символьное выполнение эквивалентно выполнению программы на наборе конкретных тестовых значений входных переменных, что сокращает мощность множества создаваемых тестов. Также определяется альтернативная семантика выполнения программы – семантика символического выполнения для языка программирования, в котором объекты данных представлены в виде символов. Семантика задается путем расширения базовых конструкций языков программирования для работы с символьными значениями.

2. Характеристики качества программного обеспечения

Одним из важных этапов верификации ПО является проверка ПО на соответствие заявленным качественным характеристикам. Наиболее важные характеристики программного обеспечения приведены ниже:

- корректность (соответствие системы своему предназначению);
- безопасность системы;
- устойчивость системы в случае недетерминированного поведения окружения (например, неверные входные данные);
- эффективность использования ресурсов времени и памяти;
- адаптируемость системы к небольшим изменениям окружения;
- переносимость и совместимость.

3. Методы верификации ПО

Существует множество методов верификации ПО, в данной статье рассматривается лишь несколько методов, приведенных на рис 1., Это, прежде всего, символьное

выполнение, проверка моделей, динамические и статические методы верификации ПО. Данные методы являются наиболее эффективными на данный момент. Изучение алгоритмов и принципов работы существующих методов верификации, послужит основой для создания синтетического метода верификации на основе SMT-решателей.

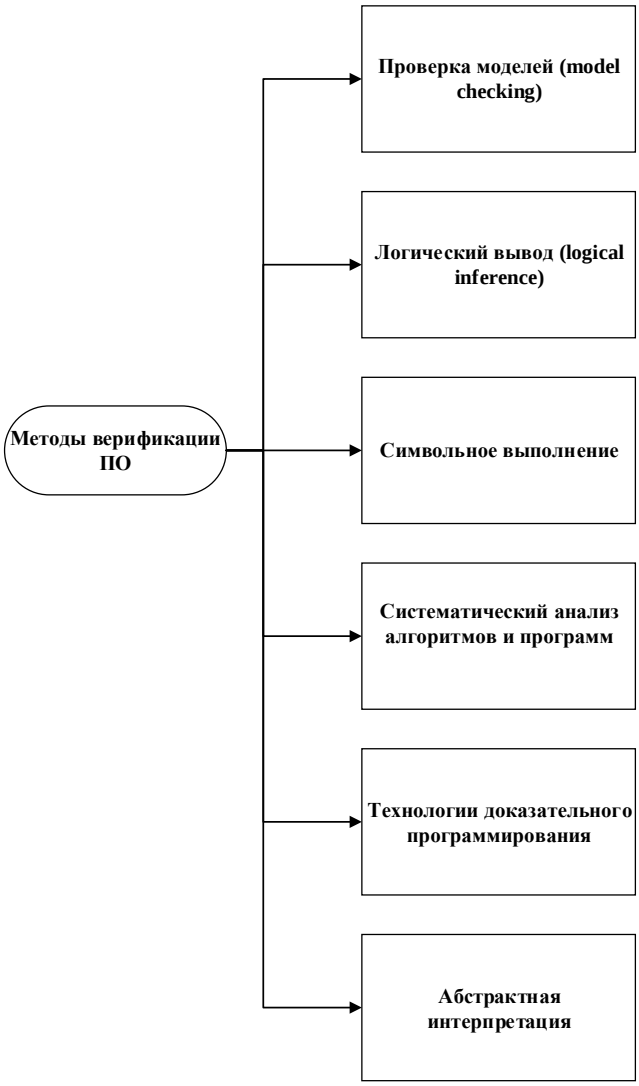

Рис. 1 – Методы верификации ПО

4. Классификация методов верификации ПО. Экспертиза. Формальные методы

Необходимо ввести ряд понятий лежащих в основе данной классификации методов.

- Виды метода – описание основных видов рассматриваемого метода;
- Уровень автоматизации – уровень, определяющий в какой степени можно автоматизировать алгоритмы того или иного метода верификации;
- Уровень функциональной пригодности – определяет насколько широкий круг задач покрывает метод верификации ПО;

- Точность – характеристика качества получаемых измерений, показывает, на сколько погрешность метода (оценка отклонения измеренного значения величины от ее истинного значения) стремится к нулю;
- Типы обнаруживаемых ошибок;
- Эффективность – определяет продуктивность метода;
- Область применимости – определяет на каком этапе разработки применим тот или иной метод верификации ПО, а также определяет к каким артефактам ПО можно применить метод верификации. Артефактом ПО – называется исследуемый участок кода программы;
- Время выполнения – определяет время, которое требуется для верификации программы;
- Способ достижения результата – определяет методы, и алгоритмы, с помощью которых инструменты анализа осуществляют верификацию ПО.

Основными видами и характеристиками являются [2], устойчивость системы в случае недетерминированного поведения окружения, эффективность использования ресурсов времени и памяти, именно по этим параметрам, как правило, проводится верификация ПО.

Одним из наиболее распространённых методом верификации ПО, является экспертиза[2]. Под которой понимается исследование программного обеспечения, проводимое лицом (экспертом), сведущем в данной предметной области, эксперт может является, как и создателем программного продукта, так и лицом или группой лиц, привлеченным со стороны, для беспристрастной оценки характеристик программного продукта.

Экспертиза может быть общей или специализированной, причем общую экспертизу можно разделить на следующие виды [2].

- **Техническая экспертиза.** Определение пригодности программного продукта для использования его по назначению, соответствие его спецификации и стандартам;
- **Сквозной контроль**. Анализ и оценка программы путем проверки артефакта, при помощи группы экспертов, участники команды последовательно представляют все характеристики программы, а эксперты анализируют ее и вносят замечания отмечая возможные ошибки и уязвимости;
- **Инспекция**. Анализ, при котором поиск ошибок и уязвимостей осуществляется в соответствии с точным планом;
- **Аудит**. Анализ программы, который выполняется людьми, не входящими в команду проекта.

В свою очередь, специализированную экспертизу можно разделить на следующие виды [2]:

- **Организационная экспертиза.** Контроль руководством состояния проекта;
- **Экспертиза удобства использования**. Контроль заказчиком и пользователем удобства использования разрабатываемого программного обеспечения;
- **Экспертиза защищенности.** Контроль специалистами по информационной безопасности защищенности использования разрабатываемого программного обеспечения;

- **Анализ свойств архитектуры.** Анализ свойств архитектуры ПО, оценка и классификация сценариев взаимодействия ПО с пользователем.

Экспертиза ПО осуществляется группой квалифицированных специалистов, причем этот метод невозможно выполнить автоматически, так как все этапы экспертизы осуществляются экспертами. Тем не менее этот метод обладает высокой функциональной пригодностью, и способен решать широкий круг задач верификации ПО, применим к любым свойствам программного обеспечения и на любом этапе верификации ПО. Точность экспертизы напрямую зависит от квалификации и опыта специалистов, проводящих ее. Исследования показывают, что от 50 до 90% ошибок и уязвимостей выявляется с помощью метода экспертизы [7]. Метод позволяет выявить практически любые виды ошибок и наиболее эффективен, в том случае если экспертизу проводят наиболее опытные и квалифицированные специалисты. Неоспоримым преимуществом, является применимость этого метода на любом этапе разработки проекта. Время выполнения проверки напрямую зависит от сложности ПО и квалифицированности команды специалистов.

Таким образом наибольшим преимуществом этого метода является его применимость на любом этапе проекта и высокая степень покрытия классов ошибок и уязвимостей.

Формальные методыверификации подразумевают под собой, верификацию математической модели программы, а не ее исходный код. Требования к программной модели формулируются в виде спецификации. Проверяется выполнимость требований спецификации на модели программы. [1]

Формальные методы можно разделить в соответствии со следующими признаками [1]:

- Дедуктивный анализ;
- Проверка моделей;
- Проверка согласованности;
- Абстрактная интерпретация.

По сравнения с экспертизой, явным преимуществом формальных методов является возможность автоматизации процесса верификации и построения моделей программ. Тем не менее для построения математической модели всегда необходим квалифицированный специалист. Формальные методы обладают высокой функциональной пригодностью, а также высокой точностью в том случае, если построена адекватная формальная модель. При помощи формальных методов можно выявить следующие классы ошибок [8]:

- Неопределенное поведения программы;
- Неинициализированные переменные; обращение к NULL указателям;
- Нарушение правил и алгоритмов пользования библиотекой;
- Сценарии приводящие к недокументированному поведению программы;
- Переполнение буфера.
- Сценарии мешающие кросс – платформенности;
- Ошибки, возникающие в повторяющемся коде;
- Ошибки форматных строк;

- Ошибки при использовании стандартных библиотек.

Недостатками методов формальной верификации является, ограниченный круг решаемых задач верификации ПО, а также то что не всегда можно построить наиболее полную и адекватную математическую модель, но при этом способны эффективно работать в промышленных проектах. Данный метод применим только к тем проверяемы участкам, которые можно учесть в формальной модели.

Для построения математических моделей, обычно используется структура Крипке [9], адля спецификации программного обеспечения используют темпоральную логику [10], автоматическое доказательство теорем, использование мультимножеств и графов, модельные подходы (конечные автоматы, сети Петри, временные автоматы, логическое описание).

Главным достоинством метода проверки моделей, является возможность автоматизации процесса верификации и построения модели. Построение формальной модели позволяет представить, код программы в виде ряды логических выражений, тем самым позволяет проверить свойства программы, выраженные в виде спецификации.

5. Статический анализ программного обеспечения

Статический анализ программы – это анализ который выполняется без фактического выполнения программы (анализ, проведенный при выполнении ПО известен как *динамический анализ*). В большинстве случаев анализируется некоторая версия исходного кода. В отличие от динамического анализа, статический анализ позволяет проанализировать все возможные пути выполнения программы. Этот термин обычно используется в случае, когда анализ производится с помощью *автоматизированных инструментов*.

На данный момент наибольшее распространение получили две группы методов статической верификации: методы дедуктивного анализа программ и методы проверки моделей [11]. Методы дедуктивного анализа используются для доказательства соответствия программы своей спецификации, обычно задаваемой в виде пред и постусловий. На текущем уровне развития эти инструменты не применимы для анализа больших программ, так как требуют ручной аннотации функций и циклов в тексте программы [13]. Методы проверки моделей (model checking) исходя из кода программы формируют её математическую модель, обычно в качестве модели используется модель Крипке [13], далее проводят анализ этой модели на предмет выполнения установленных условий и ограничений.

При верификации методом проверки модели, анализируется не сама программа, а ее математическая модель, точность и полнота анализа зависит от того, на сколько адекватной является построенная математическая модель программы.

При статической верификации происходит разбор текста программы, в ее внутреннее представление, в результате которого строится граф потока управления [11]. Генерация внутреннего представления программы происходит в процессе синтаксического анализа и позволяет сохранить ее исходную синтаксическую структур.

Вершины графа соответствуют операторам, в тексте программы, а ребра ассоциируются с передачей потока управления.

Статический анализ может быть двух видов:

- Проверка правил корректности;
- Поиск дефектов по шаблонам.

Методы статического анализа обладают высокой степенью автоматизации, это позволяет возложить задачи верификации ПО на инструменты анализа. Статический анализ обеспечивает наиболее полное покрытие кода, по сравнения с методами динамической верификации, только в том случае, если верифицируемая программа не содержит участков динамически – генерируемого кода. Методы статического анализа не зависят от используемого компилятора и среды, что позволяет находить скрытые ошибки, ошибки не определённого поведения программы, легко обнаруживает опечатки в тексте программы и ошибки, вызванные вставкой и копирования различных частей кода. Но при этом статический анализ слабо эффективен в диагностике ошибок, связанных с утечкой памяти, и имеет особенность генерировать большое количество ложных срабатываний помечая все подозрительные места в тексте программы. Тем не менее современные методы обладают высокой точностью и полнотой анализа [14]. Данный метод позволяет определять такие ошибки как:

- Неопределенное поведения программы – неинициализированные переменные;
- Обращение к NULL указателям;
- Нарушение правил и алгоритмов пользования библиотекой;
- Сценарии, приводящие к недокументированному поведению программы;
- Переполнение буфера;
- Сценарии мешающие кросс – платформенности;
- Ошибки, возникающие в повторяющемся коде;
- Ошибки форматных строк;
- Ошибки при использовании стандартных библиотек.

Верификация методом статического анализа наиболее эффективна на этапе конструирования ПО, так как статический анализ применим к исходному тексту программы и не подразумевает ее выполнения, это позволяет существенно снизить стоимость проекта и повысить его надежность. Инструменты автоматической верификации на основе статического анализа применяются достаточно широко, поскольку удобны и просты в использовании и не требуют специальной подготовки программы.

6. Характеристики статического анализа

Методы статического анализа определяются рядом характеристик, наиболее важные их них это точность и полнота анализа. При использовании инструментов, которые не удовлетворяют этим характеристикам, анализ программы может быть не полным, при анализе может возникнуть большое количество “ложных срабатываний”. Одним из способов повысить точность и полноту анализа является использование механизма зависимостей [12].

Зависимость представляет собой строго определенную связь между значениями двух и более переменных, которая может быть описана предикатом вида

$$f(x, y) = true \ .$$

Механизм зависимостей применим в случае, если текущее состояние программы объединяет два или более состояния, которые были получены на различных путях выполнения. Зависимости способны связывать различные типы программных объектов [12].

Зависимости можно разделить по характеру связи между программными объектами.

- Арифметические зависимости
- Зависимость эквивалентности
- Логические зависимости
- Зависимости сравнения

Зависимости могут быть двух видов, присваивания и слияния. Зависимости присваивания возникают в результате интерпретации определенного оператора присваивания. Зависимости слияния возникают в случае слияния двух ветвей, примером может послужить конструкция switch – case или if – else.

Реализация анализа и выявления зависимостей в рамках метода абстрактной интерпретации

Абстрактная интерпретация представляет собой метод, при котором происходит верификация динамических свойств и отсутствия ошибок при выполнении программы. Данный метод оперирует состоянием программы.Состояние программы представляет собой множество элементов абстрактного семантического домена, то есть множеством значений, которое может принимать переменная. [13, 14].

При анализе программы происходит выявление зависимостей присваивания и слияния.

Например, если$a = f(x)$и $x = g(b)$, то по окончании времени жизни x строится зависимость $a = f\big(g(b)\big)$[9- 10].

Для различных объектов программы можно сформулировать аналогичные правила.

Например, для функции языка Си seziof() можно построить зависимость возвращаемого значения функции от ее аргумента.

Результатом процесса интерпретации зависимостей является логический вывод из известных на данный момент предикатов. Логический вывод основывается на известных правилах доказательства теорем, которые можно разделить на несколько групп:

• общие правила логического вывода: Modus Ponens, подстановка;

• правила алгебры логики;

• правила арифметики целых чисел.

В будущем планируется применить для интерпретации зависимостей существующие средства логического вывода.

Для реализации методов интерпретации зависимостей, используются следующие средства логического представление на рис. 2.

Рис. 2 – Средства логического вывода

Анализ средств логического вывода, представленных на рис. 2 показывает, что наиболее эффективным инструментов являются SMT-решатели. SMT-решатели ориентированы на решение задач в логике предикат первого порядка, и включают в себя мощный аппарат разрешения систем линейных уравнений и неравенствМногие SMT-решатели используют унифицированный язык описания задач SMT-LIB, что позволяет достаточно просто заменить один решатель на другой. Остальные средства логического выводы менее эффективны, так как обладают менее мощными алгоритмами доказательства простых утверждений. Так же язык Prolog обладает недостаточной мощностью вывода для решения сложных задач.

7. Динамические методы верификации ПО

Это методы, в рамках которых анализ программного обеспечения осуществляются при помощи реального выполнения программы. В случае имитационного моделирования выполняется не сама программа, а программа ее моделирующая.

На вход программы поступают последовательности данных, которые могут вызвать недетерминированное поведение, тем самым позволяя обнаружить уязвимости и ошибки.

Динамический анализ можно разделить на несколько видов:

- Тестирование
- Мониторинг
- Имитационное тестирование
- Профилирование

Мониторинг представляет собой метод, при котором идет наблюдение, регистрация и оценка работы программного обеспечения [2]. Протоколируемая информация зависит от оцениваемых характеристик системы. Мониторинг может получать данные о работе, используя различные методы инструментированния. Инструментирование – это возможность отслеживания или установления количественных параметров уровня производительности программного продукта, а также возможность диагностировать ошибки и записывать информацию для отслеживания причин их возникновения.

Инструментирование исходного или бинарного кода может быть:

- Ручное (manual)
- Компиляторное (compiler assisted).
- На основе бинарной трансляции (binary translation).
- Инструментирование времени выполнения или инъекция времени выполнения (runtime instrumentation, runtime injection).
- Симуляторный мониторинг (simulation-based, hypervisor-based) – осуществляет протоколирование работы проверяемого ПО симулятором, на котором выполняется [1].

По способу получения характеристик ПО техники мониторинга могут быть разделены на следующие группы:

- Основанные на событиях (используют для создания знаний полную запись событий и их сопутствующих атрибутов)
- Статические (оценка системы на основе снимков ее состояния, которые получены через определенные промежутки времени)

Наиболее эффективным и всеобъемлющим методом динамического анализа является тестирование.

Тестирование программного обеспечения направлено на поиск тех ситуаций в программном коде, в которых поведение программы становится недетерминированным, не правильным и не соответствующим спецификации. Обычно тестирование осуществляется в рамках известных, заданных сценариев. На рис. 3 представлены основные виды тестирования ПО.

Обычно методы тестирования включают в себя мониторинг. Этим методы позволяют создать контролируемую среду выполнения программы, что позволяет опробовать различные наборы тесты и запротоколировать полученные результаты. Полноценное тестирование характеризуется тем насколько хорошо определены цели тестирования, обеспечена полнота тестирования и определены критерии полноты тестирования

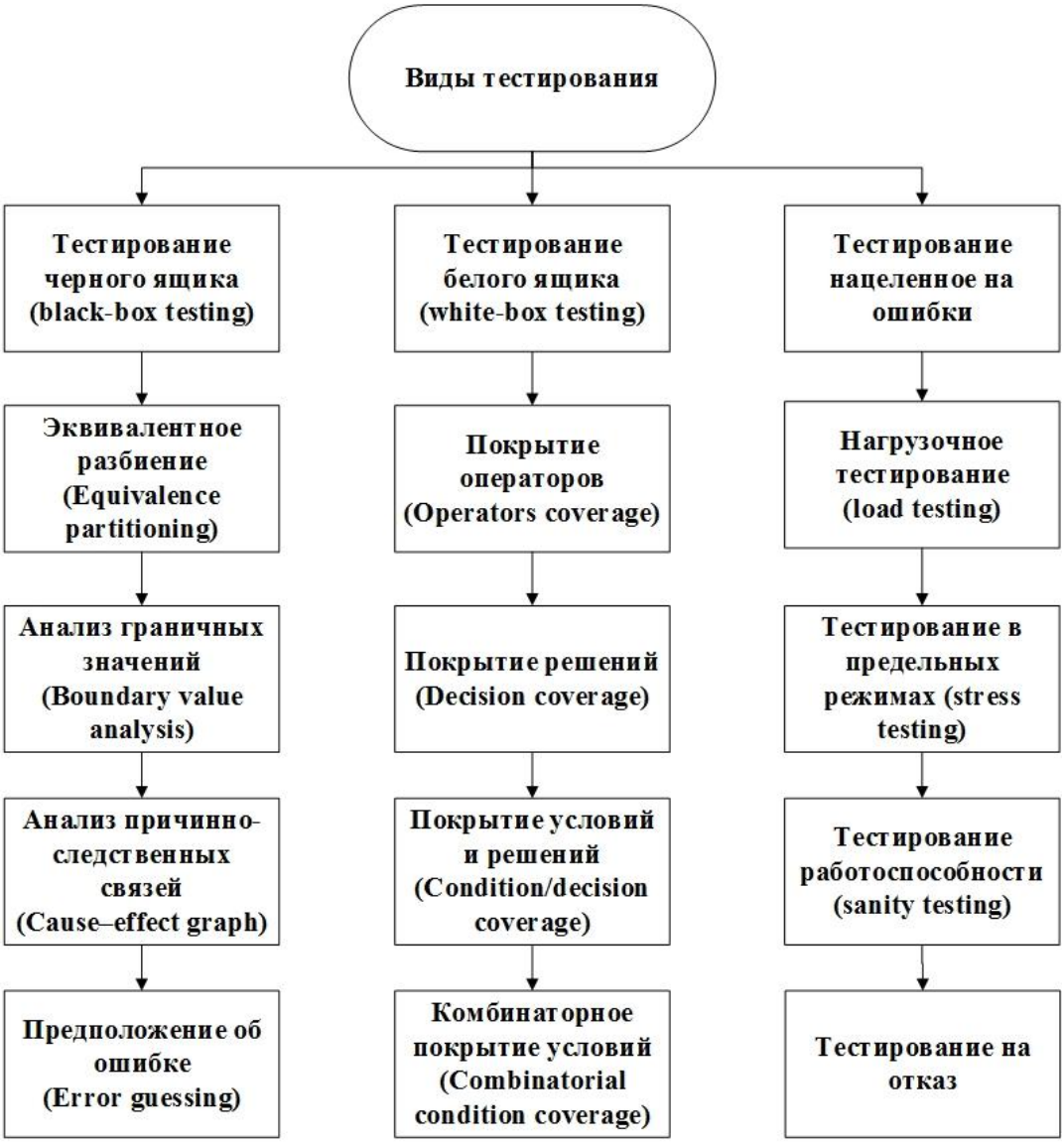

Рис. 3 – Виды тестирования

Подготовка тестов производится вручную, сам процесс тестирования и мониторинга можно автоматизировать. Методы динамического анализа позволяют провести проверку программы, в случае отсутствия исходного кода, путем создания контролируемой среды выполнения программы, а также позволяют обнаружить множество дефектов и получить наиболее точную оценку качества сложной системы. Набор тестов и систему мониторинга можно использовать многократно. В отличие от статического анализа и формальных методов, позволяет выяснить временные и количественные характеристики программного обеспечения, такие как время выполнения программы в целом и время выполнения ее отдельных участков, и количество используемых ресурсов (например: занимаемая приложением оперативная память). Динамический анализ помимо стандартного набора программных ошибок приведённого выше, позволяет определить те виды ошибок, которые возникают при исполнении программы. Динамический анализ способен выявить утечки памяти и ошибки, которые возникают в многопоточных приложениях, например, “состояние гонки” и виды ошибок, которые возникают в конечной реализации

программного продукта, так называемые “гейзенбаги” (плавающие ошибки), такие ошибки очень трудно определить на этапе конструирования, так как ошибки такого виды исчезают или меняют свойства при попытке их обнаружения. Эффективность методов динамического анализа напрямую зависит от качества и количества входных данных. Обычно эти методы применимы в тех областях где главным критерием программного обеспечения является время отклика, потребляемые ресурсы и надежность. Такими системами могут являться сервера с базами данных и системы реального времени.

8. Выводы

Разработанная классификация позволила сформулировать список требований и рекомендаций для создания синтетического метода верификации ПО. Метод представляет собой верификацию модели программы, описанную на языке SMT – Lib, ипозволяет автоматическистроить модели программы по исходному коду, с минимальным участием эксперта. Верификация модели предполагает большую степень автоматизации, но при этом допускать участие эксперта. Синтетический метода обеспечивает высокий уровень масштабируемости и высокую степень покрытия классов ошибок. Осуществлялся поиск и проверку, определенного пути выполнения программы. Верификация осуществляется за счет построения контр-примера проверяемого свойства, причем метод обеспечивает низкий уровень “шума” т.е ложных срабатываний.

Заключение

Представленная классификация является новой и позволяет на основе анализа существующих методов, составить рекомендации и требования, для создания нового, синтетического метода верификации ПО. Синтетический метод осуществляет динамическую верификацию модели ПО, построенной на языке SMT-Lib, по промежуточному представлению кода программы. Как видно из приведенной выше классификации методы экспертизы невозможно автоматизировать, но они обладают неоспоримым преимуществом за счет того что покрывают большую часть существующих ошибок покрывают. Методы формальной верификации отличаются от остальных трудоемким процессом построения формальной модели программы, хотя при этом формальные методы покрывают большой класс ошибок и легко автоматизируются. На настоящий момент статические методы верификации ПО уже не обладают высокой полнотой тестирования, хотя раньше это было их неоспоримым преимуществом, все больше программ используют динамически генерируемый код, который невозможно верифицировать статическим методами.

Динамические методы проверяют только определённый набор трасс выполнения программы и не обеспечивают должную полноту тестирования программы.

В результате проведенного исследования и классификации существующих методов, был составлен список требований для нового, синтетического метода верификации.

Предлагаемый синтетический метод должен быть автоматизирован, осуществлять комбинированный (статический и динамический) анализ, а также частично или полностью решать проблему экспоненциального роста числа состояний системы, которая присуща формальным методам верификации.

Из приведённой выше классификации видна целесообразность применения методов на различных этапах проекта проектирования ПО. При тестировании программного обеспечения методами статического анализа наиболее важно повысить точность анализа, что, в свою очередь, приводит к росту потребления ресурсов: процессорного времени, памяти. Выявление и интерпретация зависимостей присваивания и слияния между несколькими отдельными переменными позволяют повысить точность статического анализа кода.

На начальных этапах разработки методы динамического анализа целесообразно применять только в том случае, если есть какие-либо работающие элементы программного обеспечения. Динамические методы позволяют обнаруживать в ПО только ошибки, проявляющиеся при выполнении программы. Для применения динамических методов верификации ПО требуется дополнительная подготовка — создание тестов, разработка тестовой системы, позволяющей их выполнять или системы мониторинга, позволяющей контролировать определенные характеристики поведения проверяемого программного обеспечения. Возможно динамическое генерирование тестов, подобное тестирование требует гораздо больше времени. Но при этом является более эффективным и современным методом тестирования, который позволяет выявить гораздо большее количество уязвимостей в коде программы, чем при использовании статических методов верификации ПО.

Метод динамического анализа не лишен недостатков, основным недостатком данного метода является большое количество ложных срабатываний. Количество ложных срабатываний при использовании современных инструментов анализа достаточно велико и составляет, от 20 до 30% [15 - 16], но тем не менее динамический анализ является достаточно эффективным методом для проверки ПО на наличие уязвимостей.

На основе предложенной классификации был создан синтетический метод верификации ПО на основе SMT – решателя, осуществляющий построение и верификацию модели программы на языке SMT-Lib, позволившийпокрыть большее количество классов ошибок, а так же увеличило скорость и эффективность анализа кода.

Список литературы

1. Бурякова Н.А., Чернов А.В. Классификация частично формализованных и формальных моделей и методов верификации программного обеспечения // Инженерный Вестник Дона. 2010. № 4. С. 129-134.
2. Кулямин В.В. Методы верификации программного обеспечения. 2008. 117 с. // Единое окно доступа к информационным ресурсам: интернет-портал. Режим доступа: http://window.edu.ru/resource/168/56168 (дата обращения 01.09.2015).
3. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010. 560 с.
4. Pasareanu C.S., Visser W. A survey of new trends in symbolic execution for software testing and analysis // International Journal on Software Tools for Technology Transfer. 2009. Vol.11, no. 2. P. 339-353. DOI: 10.1007/s10009-009-0118-1
5. Boyer R.S., Elspas B., Levitt K.N. SELECT - a formal system for testing and debugging programs by symbolic execution // Proceedings of the International Conference on Reliable

Software, Los Angeles, California, 1975. ACM New York, NY, USA, 1975. P. 234-254. DOI: 10.1145/800027.808445
6. Лаврищева Е.М., Петрухин В.А. Методы и средства инженерии программного обеспечения: учебник. М.: МФТИ, 2006. 304 с.
7. Boehm B., Basili V. Top 10 list [software development] // IEEE Computer. 2001. Vol. 34, no. 1. P. 135-137. DOI: 10.1109/2.962984
8. Beyer D. Status report on software verification (competition summary SV-COMP 2014) // Tools and Algorithms for the Construction and Analysis of Systems / ed. by E. Ábrahám, K. Havelund. Springer Berlin Heidelberg, 2014. P. 373-388. DOI: 10.1007/978-3-642-54862-8_25 (Ser. Lecture Notes in Computer Science; vol. 8413.).
9. Вельдер С.Э., Шалыто А.А. Верификация простых автоматных программ на основе метода Model Checking // XV Международная научно-методическая конференция «Высокие интеллектуальные технологии и инновации в образовании и науке»: матер. СПб.: СПбГПУ, 2008. С. 285-288.
10. Лифшиц Ю. Верификация программ и темпоральные логики. Лекция № 3 курса «Современные задачи теоретической информатики». СПб., ИТМО, 2005. С. 3-8. Режим доступа: http://yury.name/modern/03modernnote.pdf (дата обращения 01.09.2015).
11. Мандрыкин М.У., Мутилин В.С., Новиков Е.М., Хорошилов А.В. Обзор инструментов статической верификации Си программ в применении к драйверам устройств операционной системы Linux // Сборник трудов Института системного программирования РАН. Т. 22. М.: ИСП РАН, 2012. С. 293-294. DOI: 10.15514/ISPRAS-2012-22-17
12. Глухих М.И., Ицыксон В.М., Цесько В.А. Использование зависимостей для повышения точности статического анализа программ // Моделирование и анализ информационных систем. 2011. № 4. С. 68-79.
13. Nielson F., Nielson N., Hankin C. Principles of Program Analysis. Corr. 2nd printing. Springer, 2005. 452 p.
14. Cousot P. Abstract Interpretation // ACM Computing Surveys. 1996. Vol. 28, no. 2. P. 324-328. DOI: 10.1145/234528.234740
15. Гурин Р.Е. Обзор и анализ инструментов, который осуществляют верификацию бинарного кода программы // Новые информационные технологии в автоматизированных системах: материалы 17-го научно-практического семинара. Вып. 17. М.: ИПМ им. М.В. Келдыша, 2014. С. 514-518.
16. Рудаков И.В., Гурин Р.Е., Ребриков А.В. Верификация программного обеспечения: обзор методов и характеристик // Национальная ассоциация ученых (НАУ). Ежемесячный журнал. 2014. № 3, ч. 2. С. 22-26.

Science & Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

Science and Education of the Bauman MSTU, 2015, no. 10, pp. 235–251.

DOI: **10.7463/1015.0823129**

Received: 28.04.2015
Revised: 05.09.2015

Methods of Software Verification

R.E. Gurin[1,*], I.V. Rudakov[1], A.V. Rebrikov[1]

*rg.bmstu@gmail.com

[1]Bauman Moscow State Technical University, Moscow, Russia

Keywords: analysis, static, dynamic, interpretation, symbolic execution, model checking

This article is devoted to the problem of software verification (SW). Methods of software verification designed to check the software for compliance with the stated requirements such as correctness, system security and system adaptability to small changes in the environment, portability and compatibility, etc. These are various methods both by the operation process and by the way of achieving result. The article describes the static and dynamic methods of software verification and paid attention to the method of symbolic execution. In its review of static analysis are discussed and described the deductive method, and methods for testing the model. A relevant issue of the pros and cons of a particular method is emphasized. The article considers classification of test techniques for each method. In this paper we present and analyze the characteristics and mechanisms of the static analysis of dependencies, as well as their views, which can reduce the number of false positives in situations where the current state of the program combines two or more states obtained both in different paths of execution and in working with multiple object values. Dependences connect various types of software objects: single variables, the elements of composite variables (structure fields, array elements), the size of the heap areas, the length of lines, the number of initialized array elements in the verification code using static methods. The article pays attention to the identification of dependencies within the framework of the abstract interpretation, as well as gives an overview and analysis of the inference tools.

Methods of dynamic analysis such as testing, monitoring and profiling are presented and analyzed. Also some kinds of tools are considered which can be applied to the software when using the methods of dynamic analysis. Based on the work a conclusion is drawn, which describes the most relevant problems of analysis techniques, methods of their solutions and at what stages of development it is advisable to use one or another method.

References

1. Buriakova N.A., Chernov A.V. Classification of partially formal and formal models and verification methods for software. *Inzhenernyi Vestnik Dona = Engineering journal of Don*, 2010, no. 4, pp. 129-134. (in Russian).

2. Kuliamin V.V. *Metody verifikatsii programmnogo obespecheniia* [Methods of software verification]. 2008. 117 p. Single Window Access to Information Resources: internet portal. Available at: http://window.edu.ru/resource/168/56168 , accessed 01.09.2015. (in Russian).
3. Karpov Yu.G. *MODEL CHECKING. Verifikatsiya parallel'nykh i raspredelennykh programmnykh sistem* [MODEL CHECKING. Verification of parallel and distributed software systems]. St. Petersburg, BKhV-Peterburg Publ., 2010. 560 p. (in Russian).
4. Pasareanu C.S., Visser W. A survey of new trends in symbolic execution for software testing and analysis. *International Journal on Software Tools for Technology Transfer*, 2009, vol.11, no. 2, pp. 339-353. DOI: 10.1007/s10009-009-0118-1
5. Boyer R.S., Elspas B., Levitt K.N. SELECT - a formal system for testing and debugging programs by symbolic execution. *Proceedings of the International Conference on Reliable Software*, Los Angeles, California, 1975. ACM New York, NY, USA, 1975, pp. 234-254. DOI: 10.1145/800027.808445
6. Lavrishcheva E.M., Petrukhin V.A. *Metody i sredstva inzhenerii programmnogo obespecheniya* [Methods and tools for software engineering]. Moscow, MIPT Publ., 2006. 304 p. (in Russian).
7. Boehm B., Basili V. Top 10 list [software development]. *IEEE Computer*, 2001, vol. 34, no. 1, pp. 135-137. DOI: 10.1109/2.962984
8. Beyer D. Status report on software verification (competition summary SV-COMP 2014). In: Ábrahám E., Havelund K., eds. *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2014, pp. 373-388. DOI: 10.1007/978-3-642-54862-8_25 (Ser. *Lecture Notes in Computer Science*; vol. 8413.).
9. Vel'der S.E., Shalyto A.A. Simple verification of automata-based software based on Model Checking method. *15 Mezhdunarodnaya nauchno-metodicheskaya konferentsiya «Vysokie intellektual'nye tekhnologii i innovatsii v obrazovanii i nauke»: mater*. [Proc. of the 15th International scientific-methodological conference "High intellectual technologies and innovations in education and science"]. St. Petersburg, SpbSTU Publ., 2008, pp. 285-288. (in Russian).
10. Lifshits Yu. *Verifikatsiya programm i temporal'nye logiki. Lektsiya № 3 kursa "Sovremennye zadachi teoreticheskoi informatiki"* [Software verification and temporal logic. Lecture no. 3 of the course "Modern problems of theoretical computer science"]. St. Petersburg, ITMO University Publ., 2005, pp. 3-8. Available at: http://yury.name/modern/03modernnote.pdf , accessed 01.09.2015. (in Russian).
11. Khoroshilov A.V., Mandrykin M.U., Mutilin V.S., Novikov E.M. Static Verification Tools for C Programs and Linux Device Drivers: A Survey. *Proceedings of the Institute for System Programming. Vol. 22*. Moscow, ISP of RAS, 2012, pp. 293-326. DOI: 10.15514/ISPRAS-2012-22-17 (in Russian).
12. Glukhikh M.I., Itsykson V.M., Tsesko V.A. The Use of Dependencies for Improving the Precision of Program Static Analysis. *Modelirovanie i analiz informatsionnykh system = Modeling and Analysis of Information Systems*, 2011, no. 4, pp. 68-79. (in Russian).
13. Nielson F., Nielson N., Hankin C. *Principles of Program Analysis*. Corr. 2nd printing. Springer, 2005. 452 p.

14. Cousot P. Abstract Interpretation. *ACM Computing Surveys*, 1996, vol. 28, no. 2, pp. 324-328. DOI: 10.1145/234528.234740
15. Gurin R.E. Review and analysis of tools which provide verification of the binary program. *Novye informatsionnye tekhnologii v avtomatizirovannykh sistemakh: materialy 17-go nauchno-prakticheskogo seminara. Vyp. 17* [New information technologies in automated systems: proceedings of the 17th scientific-practical seminar. Vol. 17]. Moscow, Publ. of Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, 2014, pp. 514-518. (in Russian).
16. Rudakov I.V., Gurin R.E., Rebrikov A.V. Software verification: review of methods and characteristics. *Natsional'naya assotsiatsiya uchenykh*, 2014, no. 3, pt. 2, pp. 22-26. (in Russian).