УДК 004.415.53

РЕГРЕССИВНОЕ ТЕСТИРОВАНИЕ: МЕТОДЫ И БУДУЩИЕ НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ

Г.Б. Мороз, А.В. Плис

Інститут програмних систем НАН України
03187, Київ, проспект Академіка Глушкова, 40.
тел.: 526 3309.
E–mail: mgb@isofts.kiev.ua, andry23@bigmir.net

Критическим этапом в жизненном цикле программного обеспечения является этап сопровождения, на котором предполагается поддержка командой разработчиков программного обеспечения, поставляемого их клиентам. По таким причинам как исправление ошибок, сопровождение программного обеспечения приводит к расширению его возможностей, отказу от неактуальных возможностей и оптимизации. Измененное или модифицированное программное обеспечение требует тестирования, которое известно как регрессивное тестирование. В статье представлен обзор двух основных групп методов регрессивного тестирования (методы выбора регрессивных тестов и методы приоритезации) и обсуждение открытых проблем и потенциальных направлений будущих исследований.

The most crucial phase in the software life cycle is maintenance phase, in which the development team is supposed to maintain the software which is delivered to the clients by them. Software maintenance results for the reasons like error corrections, enhancement of capabilities, deletion of obsolete capabilities and optimization. Now the changed or modified software needs testing known as regression testing. This paper surveys the two main groups of methods of regression testing (regression test selection methods and methods of prioritization) and discuss open problems and potential directions for future research

Введение

Одним из наиболее важных этапов жизненного цикла программного обеспечения (ПО) является этап сопровождения. Согласно данным, приведенным в [1], затраты на этом этапе могут превышать две трети затрат на весь жизненный цикл ПО. На этом этапе разработчику от пользователей ПО поступают сообщения об обнаруженных дефектах, запросы на расширение, улучшение или адаптирование к изменяющейся среде существующей функциональности, предложения по переносимости его на различные платформы или языки программирования и т. п. В течение некоторого времени вся эта информация интегрируется, а затем осуществляется модификация данного ПО, чтобы удовлетворить потребности пользователей. Новая версия ПО обязательно подвергается, так званому, *регрессивному тестированию* (РТ), чтобы гарантировать, что никакие новые ошибки (называемые ошибками регресса) не были введены в ранее проверенный код (т. е. неизмененные части программы) [2]. После того как РТ завершено, выпускается новая версия ПО, которая затем подвергается подобному циклу.

РТ является дорогим действием. Оно требует больших ресурсо-временных затрат, которые часто составляют почти половину всех затрат на поддержку ПО [2]. Поэтому минимизация ресурсо-временных затрат на регрессивное тестирование является очень важной как научной, так и прикладной проблемой, решение которой позволит улучшить качество и существенно сократить общие затраты на поддержку ПО. Несмотря на большое количество появившихся в последние два десятилетия публикаций, посвященных этой проблеме, она еще весьма далека от своего полного решения. Как показывают результаты некоторых исследований [3], подходы, которые чаще всего используются на практике для идентификации необходимых регрессивных тестов, основаны или на экспертной оценке, или на некоторой форме ручного анализа программы**.** Поскольку выбор тестов на основе экспертной оценки имеет тенденцию становиться все менее эффективным и надежным с увеличением размера и сложности ПО, а при ручном анализе ПО даже для умеренно сложных программ, обычно, чрезвычайно трудно вручную идентифицировать тесты, которые имеют отношение к изменениям в ПО, то в конечном итоге все это приводит к излишне высоким стоимостно-временным затратам на РТ и значительной потере тестов, которые в состоянии обнаружить ошибки регресса.

Можно констатировать, что на сегодня при наличии большого количества публикаций, посвященных различным аспектам РТ, все же наблюдается острая нехватка инструментальных средств поддержки автоматической генерации наборов регрессивных тестов, разработка которых требует дальнейших серьезных как теоретических, так и экспериментальных исследований.

В данной работе для лучшего понимания состояния дел в области РТ мы сначала рассмотрим две основные группы методов (подходов) к РТ, а затем остановимся на некоторых перспективных, по нашему мнению, направлениях будущих исследований.

1. Регрессивное тестирование: некоторые понятия

На протяжении всей работы будем придерживаться следующих обозначений: P_0 – исходная программа, P_1 – измененная версия P_0, T_0 – набор тестов, разработанных для тестирования P_0.

Регрессивное тестирование заключается в многократном использовании T_0 на P_1 и определении новых тестов, необходимых для эффективной проверки кода, а также добавленных или измененных функциональных возможностей при разработке P_1 [4]. Одним из стандартных методов регрессивного тестирования (РТ) является метод полного перетестирования, в котором все тесты из P_0 выполняются повторно для P_1. Однако из-за больших ресурсо-временных затрат, необходимых для выполнения всего набора тестов, и ограничений графика выпуска, которому часто подвергаются проекты ПО, этот метод практически неприемлем [2]. Другой простой метод РТ состоит в произвольном выборе тестов из T_0. Стоимость и время на его проведение могут быть заметно меньше чем на полное перетестирование, однако случайный выбор тестов может пропустить существенно больше ошибок регресса. Поэтому этот метод тоже не представляет особого практического интереса.

На сегодня можно выделить две основные группы методов РТ: (1) методы Выбора Регрессивных Тестов (ВРТ), (2) методы приоритезации (т. е. установления приоритетов) тестов.

Проблема выбора регрессивного теста формально может быть определена таким образом [5]:

Дано: P_0, P_1 и T_0.

Проблема: найти подмножество тестов $T_1 \subseteq T_0$, которое удовлетворяет условию: каждая обнаруженная ошибка при выполнении P_1 с набором T_0 также будет обнаружена при выполнении P_1 с набором T_1.

Следует заметить, что использование методов ВРТ может уменьшить стоимость РТ по сравнению с методом полного перетестирования только тогда, когда стоимость нахождения T_1, меньше, чем разница между стоимостями выполнения всех тестов из T_0 и T_1.

Выбор регрессивных тестов, по существу, состоит из двух основных действий:

1) *идентификация затронутых частей,* что подразумевает определение неизмененных частей программы, которые затронуты модификациями;

2) *выбор тестов*, что включает идентификацию подмножества тестов из T_0, который эффективно протестирует неизмененные части программы. Цель – выбрать подмножество тестов $T_1 \subseteq T_0$, имеющие «потенциал» для обнаружения ошибок, вызванных изменениями.

Проблема приоритезации определяется следующим образом [6]:

Дано: T_0, Ω – множество всех возможных перестановок T_0, $f : \Omega \to \mathbb{R}$ – вещественная функция, примененная к любой такой перестановке и вычисляющая ее ценность.

Проблема: найти подмножество $T_1 \in PT$, такое что $(\forall\, T_2)\,(T_2 \in \Omega)\,(T_2 \neq T_1)\,[f(T_1) \geq f(T_2)]$.

Приоритезация регрессивных тестов упорядочивает их таким образом, чтобы в процессе РТ тесты с более высоким приоритетом (относительно некоторой метрики) всегда выполнялись раньше, чем тесты с более низким приоритетом. В качестве такой метрики в большинстве случаев берется метрика APFD (**A**verage of the **P**ercentage of **F**aults **D**etected) или ее модификации [7]. Она выражается в процентах и имеет вид:

$$APDF = 1 - \frac{\sum_{i=1}^{n} TF_i}{nm} + \frac{1}{2n},$$

где n – количество тестов в наборе; m – количество ошибок, выявленных данным набором тестов; TF_i – порядковый номер теста в упорядоченном наборе, первым обнаружившим i -ю ошибку.

Чем больше ее значение, тем быстрее выявляются ошибки.

2. Методы выбора регрессивных тестов

В зависимости от парадигмы программирования методы ВРТ по разному устанавливают компромис между стоимостью выбора и выполнения тестов, с одной стороны, и эффективностью обнаружения ошибок – с другой. Далее рассмотрим подходы к ВРТ для процедурных, объектно-ориентированных, компонентно-ориентированных и аспектно-ориентированных программ, а также приложений с базами даннях, Веб-приложений и сервисов.

2.1. Методы ВРТ для процедурных программ. Различные подходы к ВРТ для таких программ можно условно сгруппировать в следующие основные классы:

1) методы на основе анализа потока данных;

2) методы, основанные на срезах;

3) методы, основанные на брандмауэре или брандмауэр-методы;

4) методы на основе различий;

5) методы, основанные на анализе потока управления.

Методы, основанные на анализе потока данных. Эти методы ВРТ обнаруживают пары «определение – использование» переменных, которые затрагиваются модификациями программы**,** и выбирают тесты, которые отрабатывают пути от определения измененных переменных до их использования. Методы ВРТ на основе потока данных, приведенные в [8], обычно выполняют анализ либо путем отработки одного изменения за другим и постепенного обновления информации потока данных в P_1, либо вычисляют полную информацию потока данных для P_0 и P_1 и сравнивают различия между парами «определение - использование». Оба эти метода требуют сохранения информации потока данных в ходе сеансов тестирования или повторного вычисления в начале каждого сеанса тестирования.

Методы, основанные на срезах. Современные технологии создания программ предполагают так называемый *slicing* — выделение из программы ее определенных частей, называемых *срезами программы*. Agrawal и другие [9] предложил ряд методов ВРТ, основанных на срезах программы. Цель этих методов состоит в выборе таких тестов, которые могут порождать разные выходы при их выполнении с измененной версией программы P_1. Авторы определяют срез теста t как набор выполняемых операторов программы, когда P_0 выполняется с t. Они предложили подходы к вычислению четырех типов срезов: исполнительный срез, динамический срез, релевантный срез и приближенно релевантный срез. Методы к ВРТ, предложенные в [9], выбирают тест t для РТ только в том случае, если срез t, вычисленный с помощью любого из четырех подходов, содержит оператор, измененный в P_1.

Брандмауэр-методы уровня модуля. Брандмауэр-метод ВРТ, впервые предложенный Leung и White [10], основан на анализе зависимостей по данным и управлению между модулями в процедурной программе. Брандмауэр определяется как совокупность всех измененных модулей в программе, а также тех модулей, которые взаимодействуют с модифицированными модулями. Брандмауэр представляет собой концептуальную границу, которая помогает ограничить количество необходимого повторного тестирования, определяя и ограничивая тестирование только теми модулями, которые были подвергнуты изменениям. Технологии брандмауэра используют *граф вызовов* для представления структуры потока управления программы.

Методы на основе различий. Эти методы ВРТ основаны на анализе различий между P_0 и P_1. Один из таких методов на основе изменения сущности кода был предложен Chen и др. в [11] для С программ. Авторы разложили элементы программы на функциональные и нефункциональные сущности кода. Сущность кода определяется либо как непосредственно исполняемая единица кода, например, функция или оператор, либо неисполнимая единица, такая как глобальная переменная или макрос. Оригинальная программа P_0 выполняется с каждым тестом $t \in T_0$. Анализируется информация о тестовом покрытии для определения набора выполнимых сущностей кода, которые исполняются каждым тестом $t \in T_0$. Для каждой функции, которая выполняется тестом t, вычисляется транзитивное замыкание глобальных переменных, макросов и т. д., на которые ссылается функция. При изменении исходной программы P_0 определяются все сущности кода, которые были изменены для того, чтобы создать программу P_1. Тесты, использующие любую из измененных сущностей, отбираются для регрессивного тестирования P_1.

Vokolos и Frankl [12, 13] предложили метод ВРТ, который основан на определении расхождения программ P_0 и P_1, а не на использовании какого-либо промежуточного представления программ. Примитивное определение текстового расхождения программ включает в себя поиск тривиальных различий между двумя версиями, таких как вставка пустых строк, комментариев и т. д. Поэтому в предложенном подходе программы перед сравнением преобразуется в *каноническую* форму [12]. Каноническая версия P_0 снабжается средствами измерения и затем выполняется с целью формирования информации тестового покрытия. Эта информация идентифицирует базисные блоки, которые выполняются каждым тестом вместо операторов программы. Канонические версии P_0 и P_1 синтаксически сравниваются для определения изменений в коде. Информация тестового покрытия затем используется для идентификации тестов, которые выполняют затронутые части кода.

Методы, основанные на анализе потока управления. В ряде работ были предложены методы ВРТ, которые анализируют модели потока управления исходной программы для выбора тестов регрессии. Так, Laski и Szermer [14] предложили технику идентификации кластера, основная концепция которой состоит в локализации изменений программы в одну или несколько областей кода, именуемых кластерами. Кластеры определяются как части кода с одним входом и одним выходом, которые изменялись от одной версии программы к другой. Техника идентификации кластера моделирует программы P_0 и P_1 как *графы потока управления* (ГПУ), обозначаемые G_0 и G_1. Определяются вершины в G_0 и G_1, которые соответствуют изменениям в коде, и набор всех таких вершин образует кластер. Техника, основанная на идентификации кластера, использует информацию о зависимости по управлению исходной и измененной процедурами для вычисления кластеров в двух графах. Как только все кластеры идентифицированы в ГПУ P_0 и P_1, каждый из них далее представляется единственной

вершиной для формирования *сокращенного* ГПУ. Анализ сокращенных графов потока основывается на предположении, что любая сложная модификация программы может быть достигнута одной из следующих трех операций: вставка кластера в код, удаление кластера или изменение функциональных возможностей кластера.

Rothermel и Harrold в предложили метод ВРТ, который основан на пересечении ГПУ исходной и измененной программ [15]. Этод метод более эффективен по сравнению с методами ВРТ на основе обхода графа, основанных на моделях графа зависимостей [5].

Ball в работе[16] предложил более точный метод ВРТ по сравнению с предложеным в [15], моделируя ГПУ G_0 для программы P_0 специальным детерминированным конечным автоматом (ДКА), который допускает набор всех возможных полных путей в G_0. Он ввел модель графа пересечений для пары ГПУ G_0 и G_1, соответствующих оригинальной и модифицированной программам, который также интерпретируется в терминах ДКА. Техника ВРТ Balla основана на достижимости дуг в графах пересечений. Она использует критерий покрытия дуг в качестве основы для анализа ВРТ.

2.2. Подходы к ВРТ для объектно-ориентированных программ. Объектно ориентированная парадигма основана на нескольких важных понятиях, таких как инкапсуляция, наследование, полиморфизм и т. д. Эти понятия обусловливают сложные отношения между различными элементами программы и усложняют анализ их зависимости. Кроме того, при объектно-ориентированной разработке приложений важное значение для ее ускорения имеют повторное использование существующих библиотек, определение классов, исполняемые программы (black-box компоненты) и т. д. Эти библиотеки и компоненты часто подвергаются независимым модификациям для исправления ошибок и наращивания функциональных возможностей, что создает дополнительные трудности при РТ объектно-ориентированных программ (ООП), использующих эти сторонние компоненты или библиотеки, так как их исходный код часто недоступен. Возникает вопрос, как эффективно выбрать регрессивные тесты, безопасные для ООП [17].

Известные подходы к ВРТ для ООП можно условно разделить на следующие три основные категории:

1) брандмауэр-подходы;

2) подходы, основанные на модели программы;

3) подходы, основанные на модели проекта.

Брандмауэр-подходы. Эти подходы призваны определить классы, подвергшиеся воздействию в измененной версии ПО. Брандмауэр может быть определен как набор всех затронутых классов, которые должны быть повторно протестированы. В этих подходах выбираются все тестовые сценарии, которые охватывают по меньшей мере один класс в брандмауэре.

Kung и другие [18] предложили брандмауэр-подход к ВРТ для С++ программ на уровне классов. Было предложено три модели для представления зависимостей между их различными элементами: диаграмма отношений объектов (ДОО), диаграмма переходов блока (ДПБ) и диаграмма состояний объектов (ДСО). ДОО является орграфом, который представляет наследование, отношения агрегирования и ассоциации и концентрирует информацию о статических зависимостях между классами. Дуга в ДОО аннотируется типом отношения (наследование, ассоциация, агрегация), которое существует между конечными вершинами, связанными с этой дугой. ДПБ представляет интерфейс и структуру управления методами класса, а также связь класса с другими классами в программе. ДСО предназначена для охвата динамического поведения классов.

Jang и другие в [19] представили брандмауэр-подход на уровня методов к ВРТ для С++ программ. В то время как в [18] и [20] в качестве единицы тестирования рассматриваются класс и оператор, в [19] в качестве единицы повторного тестирования рассматривается метод и ставится цель идентифицировать все затронутые методы. Авторы определили некоторые общие типы модификаций, возможных для С ++ программ, и для каждой модификации построили брандмауэр уровня метода, чтобы идентифицировать влияние изменений.

Подходы на основе модели программы. Rothermel и Harrold были одними из первых, кто предложил такой поход в [20]. Они разделили проблему ВРТ для ООП на две части: ВРТ прикладной программы и ВРТ измененных или производных (полученных) классов. Для ВРТ прикладной программы в подходе моделируется оригинальная программа P_0 и измененная программа P_1 с помощью моделей Графа Межпроцедурных Зависимостей Программ.

В работе [21] предложен подход ВРТ для С ++ программ, основанный на анализе представлений потока управления оригинальной и измененной программ, расширяющий поход, что дан в [15]. Так как ГПУ представляет информацию о потоке управления только единственного метода, были введены понятия графа межпроцедурного потока управления (ГМПУ) и графа потока управления класса (ГПУК), чтобы представить поток управления многофункциональных программ и объектно-ориентированных программ, соответственно. ГМПУ используется для моделирования программ, имеющих единственную точку входа, тогда как класс может иметь несколько точек входа [21]. ГПУК используется для моделей классов и состоит из индивидуальных ГПУ для всех методов класса. Учитывая модели графа для исходной и модифицированной программ, алгоритм ВРТ [21] расширяет подход, основанный на обходе графа [15], чтобы пересечь модели и выбрать соответствующие регрессивные тесты.

В работе [22] впервые был предложен безопасный подход к ВРТ для Java-программ, основанный на анализе потока управления. Их подход является адаптацией подходов к обходу графа, предложенных в [15,

21], и может работать с различными объектно-ориентированными свойствами, такими как наследование, полиморфизм, динамическое связывание и обработка исключений. Подход включает три шага: построение промежуточных представлений для исходных программ, анализ графов и определение набора опасных дуг и выбора тестов.

Необходимость объединения эффективности точных, но дорогих, подходов к ВРТ с подходами, которые работают на более высоком уровне абстракции и являются относительно неточними, привела к появлению подходов, основанных на разбиении. В работе [23] Orso и другие представили подход двухэтапного разбиения для ВРТ больших Java программ. Подход работает в два этапа: разбиение и отбор. На этапе разбиения оригинальная и измененная программы моделируются как графы отношений между классами [23]. На этапе отбора проводится более детальный анализ разбиения и строятся модели JIG (Java Interclass Graph), представляющие измененные области кода из разбиения. JIG модели затем анализируются с помощью метода ВРТ, предложенного в [15] и отбираются тесты, которые выполняют затронутые части кода.

Mansour и Statieh в [24] предложили двухэтапный подход ВРТ, предназначенный для C # программ. В их подходе к ВРТ сначала создается диаграмма затронутых классов (ДЗК), основанная на изменениях, сделанных к модифицированной программе. ДЗК представляет модификации, сделанные на уровне класса, интерфейса, Веб- или оконных сервисов и компонентов COM+. Затем используется критерий покрытия тестами, основанный на ДЗК, для выбора подмножества тестовых сценариев.

Подходы, основанные на модели проекта. Основанное на модели тестирование ПО стало очень популярным с появлением парадигмы модельно-ориентированной разработки (МОР). В парадигме МОР модель проекта обычно постепенно уточняется до получения кода. Широко распространенное использование CASE инструментов для объектно-ориентированной разработки систем обеспечивает тесную взаимосвязь между моделью проекта и его кодом. Следовательно, модели проекта могут эффективно использоваться для анализа ВРТ ООП [25]. Язык моделирования UML является стандартом ISO для представления моделей анализа и проекта ООП.

Далее приведены некоторые важные преимущества РТ, основанного на UML [25, 26]:

– *трассируемость* – легче поддерживать трассируемость между артефактами проекта и тестами, чем трассируемость между кодом и тестами. Также проще выявить изменения между различными версиями артефактов проекта по сравнению с анализом изменений между версиями кода;

– *масштабируемость* – РТ на основе кода становится очень дорогим для больших программ. Основанное на моделях тестирование сравнительно более эффективно, поскольку модель является упрощенным представлением кода;

– *языковая независимость* – различные части ПО могут быть созданы с помощью различных языков программирования. Поэтому трудно разрабатывать и реализовывать подходы ВРТ, которые могут во время выбора тестовых сценариев принимать во внимание части, созданные на различных языках программирования. Подходы ВРТ, основанные на моделях UML, помогают преодолеть этот недостаток, так как они не зависят от реализации [26]. Несколько таких подходов приведено далее.

Ali и другие [27] предложили подход к ВРТ, основанный на анализе UML-диаграмм классов и последовательности на уровне атрибутов и операций класса. Параллелизм в диаграммах последовательности отображается при помощи асинхронных сообщений и параллельных инструкций, которые не могут быть адекватно представлены традиционными моделями ГПУ. Поэтому была введена расширенная модель потока управления под названием параллельный граф потока управления (Concurrent Control Flow Graph).

Farooq и другие [28] представили основанный на моделях подход к ВРТ, использующий информацию поведенческой UML2.1-диаграммы состояний и структурной диаграммы классов для анализа выборки тестов. В процессе разработки ПО документы UML, такие как диаграммы состояний и диаграммы классов, описывающие проект и работу ПО, часто подвергаются некоторым модификациям. Модификации, сделанные в одном документе, могут также затронуть другие части ПО. Предложенный авторами подход использует информацию диаграмм измененных классов и состояний, чтобы узнать непосредственно и косвенно задетые элементы модели.

В работе [29] предложен подход к ВРТ, основанный на анализе потока управления UML-диаграмм последовательности для среды МОР. Подход включает в себя основанное на модели преобразование диаграммы последовательности в ГПУ. Трассируемость между тестами и диаграммами последовательности используется для определения того, какие элементы ГПУ выполнены каждым тестом. Затем анализируются два ГПУ, соответствующие P_0 и P_1, чтобы узнать затронутые элементы модели, и информация трассировки используется для выбора соответствующих регрессивных тестов.

Gorthi и другие [30] предложили подход к ВРТ, основанный на UML-диаграммах прецедентов. Они использует концепцию поведенческих срезов, по которой прецеденты декомпозируются на пользовательские действия, сопровождаемые некоторыми вычислениями и выходом.

Briand и другие [26]. предложили подход к ВРТ, основанный на анализе моделей проекта UML. Их подход предполагает полную отслеживаемость между моделью(моделями) проекта, кодом и тестами. Отслеживаемость между проектом и тестами помогает ассоциировать изменения в моделях проекта с тестами, которые должны быть выполнены, чтобы отработать затронутые части в проекте.

Подходы к ВРТ, основанные на спецификации. Часто сложность осуществления ВРТ состоит в том, что тестировщики не имеют доступа к моделям проекта или фактическому исходному коду. В таких случаях анализ на основе моделей или на основе кода невозможен. Эти ограничения заставили исследователей развивать подходы к ВРТ на основе спецификаций, которые обычно доступны для тестировщиков. В этом контексте нужно отметить, что, хотя мы классифицировали эти подходы как подтип объектно-ориентированных подходов к ВРТ, они могут быть распространены на более широкий спектр парадигм программирования, таких как компонентно-ориентированное.

Chen и другие [31] предложили основанный на спецификации подход к ВРТ, который использует UML диаграммы деятельности для моделирования потенциально затрагиваемых требований и поведение системы. Они также классифицировали регрессивные тесты, подлежащие отбору, на целевые тесты и тесты безопасности. Идея состоит в том, чтобы более тщательно проверить те части кода, для которых вероятность присутствующей ошибки и ее стоимость (то есть, последствие воздействия) высока.

Chittimalli и Harrold [32] предложили подход к ВРТ, который, по существу, основан на отслеживании того, какие спецификации проверяются и какими тестами из T_0. Эта информация представляется в виде матрицы *покрытия требований* между набором требований и тестами. Подход, предложенный в [23], использовался для идентификации задетых частей кода, а также набора требований, которые затронуты вследствие изменений.

2.3. Подходы к ВРТ для компонентно-ориентированного ПО. В модели разработки компонентно-ориентированного ПО (КОПО) программный продукт создается путем интеграции различных компонентов собственной разработки или сторонних поставщиков. Надежность КОПО в значительной степени зависит от надежности отдельных компонентов. Эти компоненты, представляющие "черный ящик", часто модифицируются поставщиками с целью исправления ошибок и включения расширений. Поэтому при РТ КОПО должно определяться, каким образом изменения, внесенные в компоненты, могут повлиять на выполнение прикладных программ, использущих эти измененные компоненты. Подходы к ВРТ для традиционных программ не могут обоснованно применяться для ВРТ ПО, использующего готовые коммерческие компоненты потому, что код этих компонентов, как правило, не доступен. Остановимся кратко на основных методах ВРТ для КОПО.

Методы ВРТ на основе метаконтента. Трудности неадекватного информационного обмена между *пользователем компонента* (К-пользователем) и *разработчиком компонента* (К-разработчиком) во время разработки КОПО могут быть преодолены путем совместного использования релевантной информации о компоненте, необходимой для анализа ВРТ. Orso и другие [33] предложили концепцию информации об измененном содержимом, называемую *метаконтентом компонента*, как средства совместного использования информации об изменениях, которым компонент подвергается от версии к версии. Различные подходы к ВРТ могут определять свои собственные наборы необходимых метаконтентов, которые должны быть разделены с К-разработчиками. Ниже мы обсуждаем различные подходы ВРТ, основанные на метаконтенте.

Orso A. в [34] предложен подход к ВРТ на основе покрытия кода для КОПО и существующих процедурных подходов к ВРТ [11, 15]. В случае, если К-пользователи не знают о компонентах, которые претерпели изменения, тогда во время ВРТ прикладного кода для РТ отбирается любой тест, в котором вызывается метод модифицированных компонентов. Это может привести к выбору тестов, не связанных с конкретными изменениями. С информацией об изменениях, внесенных в компоненты, можно сделать более точный выбор тестовых сценариев.

Мао и другие обнаружили [35], что применимость подхода, предложенного в [33, 34], ограничена из-за того, что он нуждается в очень подробной метаконтентной информации, предоставляемой К-разработчиком. Они предложили подходы ВРТ [35, 36], которые акцентируют внимание на *доступности* для К-пользователей определенных данных от К-разработчиков.

Подход, основанный на встроенных тест-скриптах. О подходе к ВРТ для КОПО, который использует иной уровень информационного обмена между К-пользователями и К-разработчиками, сообщалось в [36]. Появление подхода обусловлено тем фактом, что только К-разработчики имеют подробные знания о работе компонента и модификациях, подготовленных для каждой из его версий. Этот подход предлагает, чтобы К-разработчики помещали тестовые скрипты в исходный код компонента во время модификации. Цель этих тестовых скриптов заключается в сборе информации о шаблоне выполнения компонента во время исполнения тестов. Эта информация помогает находить тесты, которые покрывают измененные операторы компонента.

Подходы к ВРТ, основанные на моделях. Предложенные для КОПО подходы к ВРТ, основанные на моделях, являются по существу развитием подходов к ВРТ, основанных на моделях, предложенных для процедурных и ООП. Ниже мы кратко обсуждаем несколько таких подходов.

В работе [37] предложен подход к ВРТ, использующий модели UML и OCL (Object Constraint Language). Предполагается, что функциональные возможности, предоставляемые измененным компонентом, являются сверхнабором функциональных возможностей, предоставленных оригинальным компонентом, то есть, новая версия компонента может включать исправления ошибок и оптимизацию существующих функциональных возможностей наряду с новыми функциональными возможностями, которые были введены.

Wu и Offutt в [38] предложили другой подход к ВРТ, основанный на моделях UML, в котором диаграммы кооперации и последовательности используются для анализа поведения потока управления компонента и того, каким образом объекты взаимодействуют друг с другом через описание сообщений. Изменения, внесен-

ные в модифицированную версию компонента, будут отражаться в диаграмме кооперации как изменения в методе класса или изменения в последовательностях взаимодействия.

В работе [39] предложен подход к ВРТ для КОПО, в котором с использованием моделей анализируется динамическое поведения (например, взаимодействие методов во время выполнения) компонентов для выбора тестовых сценариев. В этом подходе есть возможность выбрать регрессивные тесты для компонентов, разработанных в .NET и Java.

Анализ исполняемого кода. Zheng и другие [40] предложили семейство подходов к ВРТ, основанных на анализе исполняемого кода (двоичных файлов, таких как .dll. или lib) измененных компонентов. Эти подходы известны как I-BACCI (Integrated-Black-box Approach for Component Change Identification, *Интегрированный подход "черного ящика" для идентификации изменения компонентов*), с указанием номера версии для конкретизации подхода семейства. Семейство *I-BACCI* использует подход брандмауэра для анализа связующего кода (прикладного кода, который интегрирует готовые коммерческие компоненты).

2.4. Подходы к ВРТ для аспектно-ориентированных программ. Парадигма аспектно-ориентированной разработки ПО является относительно новой и направлена на улучшение разбиения ПО на модули, изолируя низкоприоритетную и вспомогательную функциональность от основной и значимой для приложения бизнес-логики. В традиционном программировании программисту свойственно иметь дело со вспомогательными функциями и располагать их чередующимися пластами (называемыми слоями функциональности, *concerns*) в основном коде приложения. Слои, которые распространяются через несколько модулей, называются сквозными (пересекающими), образуя сквозную функциональность (*crosscutting concerns).* Аспектно-ориентированное программирование (АОП) позволяет программистам отсылать эту вторичную сквозную функциональность к автономным модулям, называемым *аспектами.* АОП усвоено многими объектно-ориентированными языками программирования, а АОП-языки, такие как AspectJ, приобрели большую популярность среди Java разработчиков**.** Введение аспектов обычно изменяет поведение исходной Java программы. Таким образом, программы на AspectJ также должны быть подвержены тщательному РТ после внесения изменений. Ниже мы обсуждаем предлагаемые подходы к ВРТ для AspectJ программ.

ВРТ программ на AspectJ с использованием моделей потока управления. Zhao и другие [41] предложили подход ВРТ для AspectJ-программ, расширяя работу Harrold и др. [22]. Они предложили граф потока управления системой (ГПУС) и Граф Потока Управления Аспектом для моделирования AspectJ программ. После того как построены графы ГПУС для оригинальной и измененной пары AspectJ программ, используется метод поиска вглубь (depth-first search), предложенный в [22], для идентификации опасных дуг в графе. Тестовые сценарии, которые выполняют опасные дуги, отбираются для РТ.

Подход к ВРТ для AspectJ-программ, основанный на расширенном JIG. В работе [42] приведен подход безопасного ВРТ, который основан на представлении вспомогательного графа для AspectJ программ. Авторы сначала предложили модель графа, основанного на потоке управления для AspectJ-программ, назвав его AspectJ межмодульный граф *(*AspectJ Intermodule Graph, *AJIG*), который является расширением межклассового графа Java (*JIG,* Java Interclass Graph). Соответствующие регрессивные тесты выбираются путем сравнения графов *AJIG* для P_0 и P_1. Авторы расширили алгоритм обхода графа, предложенный в [22]. Их двухфазный алгоритм также обрабатывает ситуации, когда вершины назначения для пары сравниваемых дуг являются следами описания оператора.

2.5. Подходы к ВРТ для приложений с базами даннях. В настоящее время используется большое количество приложений с базами данных (БД), которые обычно состоят из нескольких компонентов, способствующих повышению их сложности [43]. Приложения с БД также должны часто изменяться из-за различных требований, например, изменений в компонентах, растущего количества пользователей и данных и т. д. Поэтому РТ приложений с БД является еще одним важным видом деятельности. Требования и проблемы при выборе регрессивных тестов для приложений с БД отличаются от таковых для классов программ, которые мы обсуждали до сих пор. При выборе регрессивных тестов для приложений с БД нужно учитывать следующие особенности [43, 44]:

– подходы к ВРТ для других классов программ неявно предполагают, что тесты независимы друг от друга и могут быть выполнены в любом порядке. Это предположение неверно для приложений с БД, поскольку выход теста может изменить *состояние базы данных*, влияя на выполнение других тестов. Поэтому помимо глобального состояния программы, для приложений с БД необходимо рассматривать состояния базы данных в ходе ВРТ;

– в процессе РТ состояние базы данных обычно приходится многократно сбрасывать, то есть, восстанавливать начальную конфигурацию БД. Переустановка БД является дорогостоящей деятельностью как с точки зрения затрат, так и времени;

– языки БД поддерживают такие возможности как структурированные запросы, ограничения целостности, обработка исключений и триггеры таблиц, которые усложняют анализ влияния измененных частей программы. Например, срабатывание триггеров может создать неявные зависимости межмодульного управления.

Традиционные понятия безопасности и зависимостей не могут быть применены в РТ приложений с БД, поскольку соответствующие подходы были разработаны для простых приложений. В этой связи для приложений с БД были предложены несколько подходов к ВРТ, которые мы кратко рассмотрим.

Двухэтапный подход ВРТ для систем, основанных на SQL. В этом подходе, предложенном Haraty и др. [44], помимо традиционных зависимостей по управлению и данным между элементами в приложении с БД, определяются следующие аспекты, которые необходимо учитывать:

–*зависимости потока данных* – зависимости потока данных могут возникать между модулями базы данных за счет использования сквозных таблиц модулей**;**

–*зависимости компонентов* – они возникают между различными модулями БД из-за срабатывания триггеров таблиц, изменений в таблицах или представлениях, а также вследствие изменения инструкций SQL;

–*обработка исключений (особых случаев)* **–** возникновение исключительных состояний может повлиять на отношения в потоке управления, которые должны учитываться в ходе анализа ВРТ**.**

Подход безопасного ВРТ, основанный на ГПУ. Willmor и Embury [45] развили алгоритм ВРТ, основанный на безопасном анализе потока управления, предложенном Rothermel и Harrold [15] для процедурных программ, применительно к приложениям с БД. ВРТ, основанный только на отношениях определение-использование данных, не безопасен для приложений с БД, поскольку возможна ситуация, когда инструкция запишет в БД какие-то данные, которые позже будут прочитаны оператором программы, обрабатывающим более раннюю инструкцию в каком-то пути исполнения [45]. Поэтому авторы ввели понятие *зависимостей базы данных,* чтобы охватить дополнительные зависимости, которые возникают между элементами в программе базы данных.

2.6. Подходы к ВРТ для Веб-приложений и сервисов. Веб-приложения и сервисы являются динамичными по своей природе и постоянно развиваются. Они часто обновляются и поэтому должны быть подвергнуты РТ для проверки правильности неизмененных функциональных возможностей. Далее мы обсуждаем главные особенности Веб-приложений и сервисов, которые должны быть приняты во внимание во время ВРТ:

– Веб-приложения состоят из сервисов (услуг) на стороне сервера, пользовательских приложений и промежуточного ПО (прослойки). Подход к безопасному ВРТ для Веб-приложений должен рассматривать все типы зависимостей, которые могут возникнуть в различных слоях тестируемого Веб-приложения;

– Веб-приложения и сервисы являются, по природе своей, распределенными и слабо связанными;

– Веб-сервисы обычно используют другие сервисы. Поэтому зависимости, возникающие вследствие изменения в других сервисах, также должны рассматриваться в ходе ВРТ.

Далее приводится несколько подходов к ВРТ для Веб-приложений.

Подход к ВРТ для Web приложений, основанный на срезах. Такой подход предложили Xu и другие в [46]. Они предполагали, что Веб-приложения состоят из множества статических HTML-страниц и программ, выполняемых на стороне сервера. Типы изменений, которым может подвергнуться страница HTML, могут быть разделены на следующие основные классы: вставка элемента страницы (например, якорь, гиперссылка и т. д.), удаление элемента страницы, вставка страницы и удаление страницы. Более сложные изменения декомпозируются на комбинацию из этих базовых модификаций.

Подход к ВРТ, основанный на моделях системы. Tarhini и другие [47]. предложили подход безопасного ВРТ для приложений на основе Веб-сервисов. В подходе Веб-сервисы определяются как автономные компонентные приложения, которые находятся в разных местах и имеют коммуникацию с помощью XML-сообщений, используя интерфейсы **SOAP**. Коммуникация, использующая обмен сообщениями, может также быть ограничена во времени. Услуги, предоставляемые Веб-сервисом, являются коллективно используемыми с помощью спецификаций WSDL.

Подходы к ВРТ, основанные на анализе моделей потока управления. Подход к ВРТ, предложенный Ruth и другие [48], является подходом серого-ящика (gray-box), так как сложно провести РТ белого ящика (white-box) для Веб-сервисов из-за того, что зачастую исходный код компонентов не может быть доступным разработчикам Веб-сервисов. Это подход серого-ящика, поскольку он не требует наличия исходного кода Веб-сервисов. Вместо этого предполагается, что поставщики компонентных Веб-сервисов смогут предоставить следующую информацию в виде метаданных наряду с новым вариантом сервиса: спецификация WSDL, набор тестов, ГПУ для Веб-сервисов и данные о покрытии тестами.

Lin и другие [49] предложили подход безопасного ВРТ для Веб-сервисов Java на основе преобразования кода. В их подходе моделируется Java-код на стороне клиента и на стороне сервиса как единая объединенная программа. Услуги и интерфейсы, предоставляемые Веб-сервисом, доступны из спецификаций WSDL.

3. Методы приоритезации

Чтобы лучше понять динамику исследований в области приоритезации регрессивных тестов, было, с определенной долей условности, выделено шесть основных категорий методов и подходов, которые кратко описаны ниже.

Подход, основанный на покрытии. Приоритезация на основе покрытия базируется на том факте, что чем большее покрытие достигнуто тестовым набором, тем больше шансов обнаружить ошибки на ранних стадиях процесса тестирования. В работе [50] Rothermel и др. продемонстрировали четыре метода на основе покрытия: общее/дополнительное, операторов/ветвей, соответственно. При сравнении этих методов использовался инструмент для анализа программ «Аристотель» [51], а результаты были измерены с помощью метрики APFD. Упорядочение тестового набора проводилось по степени способности к быстрому обнаружению ошибок. При сравнении методов было установлено, что приоритезация по общему покрытию выигрывает у метода приоритезации по дополнительному покрытию. Эта работа была расширена Elbaum и др. в [6] для заданной версии программы. Авторы предложили восемь методов, из которых «общая функциональность» и «дополнительная функциональность» были основаны на покрытии. При этом скорость обнаружения ошибок улучшилась. В результате сравнения всех 12 методов (4 на уровне операторов и 8 на уровне функций) худшим признан метод fn-total, лучшим - fn-fi-fep-addtl.

Srivastava и Thigarajan [52] предложили подход к приоритезации на основе двоичного кода. Была построена тестовая система *Echelon* для расположения по приоритетам множества ошибок на основе изменений, внесенных в программу. Предлагаемое преимущество использования двоичного представления состоит в устранении шага перекомпиляции для покрытия набора ошибок и т. д., что облегчает интеграцию процесса встраивания тестовой системы в производственную среду.

Приоритезация на основе графовой модели Веб-системы ISELTA с использованием подхода нечеткой кластеризации была предложена Belli и другие в [53]. Показано, что этот подход полезен, когда наборы тестов упорядочиваются в рамках ограниченного времени и способа. Влияние ограничения времени на стоимость РТ были изучены Do и другие [54]. Они предложили четыре метода приоритезации, два из которых были основаны на общем/дополнительном покрытии и два на сетях Байеса. Метод с дополнительным покрытием оказался более эффективным, чем с общим.

Jiang и другие [55] предложили девять основанных на покрытии методов приоритезации тестов при адаптивном случайном тестировании (*Adaptive Random Test*, ART). Эти методы были разделены на три группы, а именно: maxmin, maxavg и maxmax. Для каждой группы уровень покрытия данных основывался на операторах, функциях и ветвях. Сравнение предлагаемых методов из случайным упорядочением показало, что ART методы являются более эффективными, чем случайное упорядочение. Также были найдены лучший и худший методы среди всей группы ART методов.

В работе [56] Bryce и другие рассмотрен основанный на покрытии подход, касающийся разработки единой абстрактной модели тестирования путем комбинирования GUI (интерфейса пользователя) и Веб-приложений. Определение приоритетов было выполнено на основе покрытия взаимодействующих значений параметров или частоты. Был также определен обобщенный критерий приоритезации как для GUI, так и для Веб-приложений.

Подход, основанный на изменении. Цель этого подхода состоит в установлении приоритетов тестов на основе изменений, сделанных в программе. Впервые этод подход был предложен Wong и другие [57]. Korel и другие [58] предложили методы избирательной приоритезации тестов на основе системной модели и приоритезации тестов на основе модельной зависимости с использованием расширенных конечных *автоматов* для системных моделей. Хотя последний метод был более дорогим, улучшение в эффективности приоритезации наблюдалось при использовании метрик скорости обнаружения ошибки для обоих методов. В работе [59] Korel и другие предложили еще пять эвристических методов и сравнили все семь методов. Метод, основанный на модельной зависимости, показал лучшую эффективность из всех семи методов.

Основанный на модели подход к приоритезации тестов, опирающийся на связи трассируемости между моделями, тестами и артефактами кода, был предложен Filho и др. в [60]. Этот метод поддерживает РТ, основанное на изменениях, используя временные метки и приоритезацию на основе свойств**.** Авторы выполняли приоритезацию и фильтрацию как часть процесса генерации тестов, использующего модификаторы набора тестовых сценариев.

Подход, основанный на ошибках. Методы приоритезации на основе ошибок впервые были предложены в [50]. Согласно этому подходу свойство ошибки быть выявленной тестом зависит не только от того, будет ли протестирован определенный оператор, но также от вероятности того, что ошибка в операторе повлечет за собой отказ этого теста. В исследовании были представлены два метода, касающиеся потенциала теста в выявлении ошибки – метод приоритезации с учетом *FEP* (Fault Exposing Potential – Потенциал выявления ошибки) при общем покрытии и метод приоритезация по FEP при дополнительном покрытии.

Elbaum и другие в [6] представили шесть методов функционального уровня для определения приоритетности тестов относительно ошибок. Два из них являются методами приоритезации функционального уровня с учетом FEP; два других основаны на индексе ошибок (fault index, FI), который представляет собой склонность к содержанию ошибок в соответствующей функции, и еще два объединяют как FI, так и FEP, сперва применяя общую приоритезацию индекса ошибок ко всем тестам, а затем, в качестве вторичного упорядочивания, приоритезацию FEP к тем из них, которые обладают равными значениями FI. Было предоставлено достаточно статистических доказательств, чтобы показать, что методы функционального уровня менее эффективны, чем методы уровня оператора.

В дополнение к вышеупомянутым методам в [7] представлено еще четыре метода приоритезации на уровне функций, которые являются *DIFF*-базируемыми методами. Эти методы требуют вычисления синтаксических различий между двумя версиями программы. Степень изменения измеряется для каждой функции, присутствующей в обеих версиях, с помощью количества строк, добавленных, удаленных или измененных в выводе команды UNIX diff, примененной к обеим версиям**.** Два из этих четырех методов основаны только на DIFF, а два других сочетают *DIFF* с *FEP*.

Подход на основе требований. Srikant и другие [61] был предложен метод системного уровня PORT V 1.0 (Prioritization Of Requirements for Testing) для приоритезации на основе требований и разработан инструмент для его реализации. Подход основан на четырех факторах: назначенных потребителем приоритетов требований, сложности реализации с точки зрения разработчика, изменчивости требований и предрасположенности требований к ошибкам. Цель состоит в раннем выявлении серьезных ошибок и улучшении качества ПО с точки зрения потребителя. Более серьезными считались ошибки в требованиях с большим диапазоном PFV, где PFV – значение коэффициента приоритетов для конкретного требования, вычисленное по формуле, предложенной авторами. Исследование показало, что метод PORT может повысить эффективность тестирования благодаря концентрации внимания на наиболее значимой для потребителя функциональности и на улучшении скорости обнаружения серьезной ошибки, тем самым минимизируя область появления ошибок.

Krishnamoorthi и другие в [62] для повышения удовлетворенности пользователя и скорости обнаружения серьезной ошибки была представлена модель приоритезации тестов для системного уровня исходя из спецификации требований к ПО. Модель расположила по приоритетам тесты системы, основываясь на следующих шести факторах: приоритет потребителя, изменения в требованиях, сложность реализации, удобство и простота использования, поток управления в приложении и последствия ошибки. Другой метод тех же авторов был представлен в [63]. Он отличался только по двум факторам, затрагивающим алгоритм установления приоритетов. Факторы, представленные в [63]: назначенный потребителем приоритет, изменения в требованиях, сложность реализации кода с точки зрения разработчика, последствия ошибки, полнота и трассируемость. При сравнении методов с общим покрытием операторов и общим покрытием методов было установлено, что для метода авторов скорость обнаружения серьезных ошибок возрасла.

Подход, базирующийся на прошлом опыте. Kim i Porter [64] предложили метод установления приоритетов, основанный на исторических данных выполнения программы. Они показывают, что историческая информация может быть полезной для сокращения затрат и увеличения эффективности процесса РТ. Для этого ими было введено понятие РТ с полной памятью.

Fazlalizadeh и другие [65] усовершенствовали базируемый на прошлом метод приоритезации, предложенный в [64] для ускорения обнаружения ошибок в средах с ограниченным ресурсом и временем. В работе представлено уравнение, которое учитывает историческую эффективность тестов в обнаружении ошибки, историю выполнения тестов и последний приоритет, присвоенный тестам.

Подход, на основе метаэвристик глобальной оптимизации. Метод установления приоритетов с учетом времени, использующий генетический подход, был предложен Walcott и другие [66]. Эксперимент проводился на уровне детализации программ двух объектов: Gradebook и JDepend. Были задействованы инструменты отслеживания процесса Emma и Linux и результаты были определены количественно, используя метрику APFD. В конечном счете, осуществление приоритезации с помощью генетического алгоритма (ГА) дало улучшение по сравнению с неупорядочиванием, обратным упорядочиванием и установлением приоритетов с учетом ошибок.

Другой вид приоритезации тестов на основе ГА был предложен Conrad и другие в [67]. Работа представила широкий спектр мутаций, кроссинговеров, селекций и операторов преобразования, которые использовались для изменения порядка тестов. Экспериментальное исследование было проведено на 8 реальных приложениях (таких же, как в [68]), используя одну и ту же метрику эффективности покрытия [49] и их тесты JUnit на уровне системы. Результаты были проанализированы с помощью горохообразных (beanplots) диаграмм, являющихся альтернативой диаграммам размаха. При сравнении предлагаемого метода с методами случайного поиска и поиска экстремума ГА дал лучшие результаты.

Заключение и направления будущих исследований

Признано, что подходы к РТ, которые анализируют изменения программы на более низком уровне детализации (например, на уровне операторов), более точны, чем подходы, которые выполняют анализ на сравнительно более высоком уровне абстракции (например, моделях проекта). Однако, объемные вычисления для низкоуровневого анализа делают эти подходы более дорогими и менее эффективными по сравнению с высокоуровневыми подходами. Поэтому при выборе подходящего метода ВРТ, необходимо придерживаться компромисса между *стоимостью и эффективностью,* с одной стороны, и *уровнем детализации*, с другой.

Современные коммерческие программные продукты становятся все более крупными и сложными, и для их проверки, обычно, требуется тысячи тестов. Поэтому для получения дополнительной экономии усилий при РТ исследователи должны помнить:

– с тенденцией увеличения размера приложений подходы к ВРТ должны масштабироваться для очень больших программ, имеющих размеры кода порядка миллионов KLOC. Для современных больших программных систем масштабируемость является важной проблемой. Поэтому интересным направлением исследования могло бы быть изучение композиционных и интеграционных подходов к ВРТ;

– подход к ВРТ должен учитывать при выборе тестов все возможные взаимосвязи в зависимости от рассматриваемого класса программ, то есть, он должен быть безопасным для этого класса программ.

Основанное на модели РТ. Учитывая тот факт, что статический анализ больших программных систем является в вычислительном отношении дорогостоящим, перспективными представляются подходы к ВРТ, основанные на модели, поскольку они не только хорошо масштабируются, но являются более эффективными. Кроме того, в последнее время МОР уделяется много внимания. В МОР существует тесная взаимосвязь между моделью (моделями) проекта и кодом в том смысле, что любое изменение в модели получает отражение в коде и наоборот. Поэтому вместо осуществления ВРТ на коде, его можно автоматически выполнять, базируясь на моделях проекта. Основанный на модели ВРТ может также помочь учесть некоторые аспекты поведения программы (например, переходы состояний, пути сообщений, критичность задач и т. д.), которые нелегко идентифицируются при статическом анализе кода.

*Улучшенная инструментальная поддержка РТ***.** В будущих работах по РТ следует постепенно переходить от теоретических исследований к инструментам реализации. В ряде исследований [3] отмечалось, что текущая инструментальная поддержка для автоматизации ВРТ довольно слабая. Поэтому совместные усилия должны быть направлены на разработку интегрированных инструментов ВРТ, используя механизмы записи и воспроизведения (capture-and-replay).

*Синтезированные методы РТ***.** Большинство упоминаемых в литературе подходов к РТ основаны либо на коде, либо на модели. Так как у обоих этих подходов есть свои собственные уникальные преимущества, возможно эти подходы могут быть обоснованно синтезированы, и эта проблема заслуживает дальнейшего исследования. Например, осуществляемый анализ в методе ВРТ, основанном на коде, может быть сделан более эффективным при использовании информации, доступной по UML-моделям проекта, документам Спецификации требований к ПО и т. д.

Большая часть работ, посвященных многоцелевому РТ, находится в областях минимизации набора тестов и установления приоритетов [69]. Однако в реальном РТ при выборе тестов, необходимо учитывать множество целевых показателей, таких как количество выполненных тестов, затраты, привлеченные для тестирования, достигнутое покрытие кода, время, доступное для тестирования и т. д. Интересный путь исследований мог бы состоять в объединении методов выбора регрессивных тестов с подходами либо минимизации, либо приоритезации. При таком обобщенном подходе техника выбора тестов могла бы затем включать дальнейшую минимизацию/приоритезацию. РТ, используя такой синтезированный подход, может учесть одновременно несколько целевых критериев в ходе тестирования и потенциально может помочь добиться дальнейшей экономии усилий РТ, не ставя под угрозу тщательность тестирования.

Встроенное ПО. Возрастающее использование встроенного ПО реального времени в критических по безопасности приложениях (ПО атомных станций, космических кораблей, самолетов и т. п.) привело к повышенным требованиям к качеству их кода. Высокая стоимость и сложность выполнения РТ для такого ПО должны были бы стать дополнительным стимулом к разработке особых подходов к РТ для этих программ. Однако к настоящему времени успехи в этом направлении весьма скромны, несмотря на то, что это направление исследований представляется весьма важным.

Приложения с гибридным управлением. Для приложений с дискретным управлением обычно используют UML-модели, тогда как для приложений с гибридным управлением популярны модели MATLAB Simulink/Stateflow [70]. В этом контексте необходимы подходящие подходы к ВРТ для приложений с гибридным управлением. Кроме того, более детальное исследование требуется для изучения эффективности подходов к ВРТ для тестирования нефункциональных требований.

Агентно-ориентированное программирование. В последние годы стремительно развивается агентно-ориентированная парадигма ПО [71], которую можно рассмотреть как естественное расширение объектно-ориентированной парадигмы. Сегодня программные агенты являются ключевой технологией для удовлетворения потребностей современного бизнеса. Они предлагают также эффективную концептуальную методологию для разработки адаптивных и динамичных сложных систем. На практике исследования по разработке программных агентов и мультиагентных систем приобрели очень больших масштабов и используются в различных активных областях. Однако, поскольку агенты имеют более высокий уровень абстракции по сравнению с объектами и во многих аспектах существенно отличаются от них, то известные методы РТ для ООП совершенно непреемлемы для агентно-ориентированых программ (АОП). Разработка эффективных методов РТ для АОП – еще одна очень важная область исследований.

1. *Pressman R.* Software Engineering: A Practitioner's Approach. New York: McGraw-Hill, – 2010. – 895 p.
2. *Leung H., White L.* Insights into regression testing // Proceedings of the Conference on Software Maintenance, – 1989. – P. 60–69.
3. *Grindal M., Offutt J., Mellin J.* On the testing maturity of software producing organizations // In TAIC-PART: Proceedings of the Testing: Academic & Industrial Conference on Practice And Research Techniques. – 2006. – P. 171–180.

4. *Elbaum S., Kallakuri P., Malishevsky A. G., Rothermel G., Kanduri S.* Understanding the Effects of Changes on the Cost- Effectiveness of Regression Testing Techniques // Journal of Software Testing, Verification, and Reliability. – 2003. – Vol. 13, N 2 – P. 65–83.
5. *Rothermel G., Harrold M.* Selecting tests and identifying test coverage requirements for modified software // Proceedings of the International Symposium on Software Testing and Analysis. – 1994. – P. 169–184.
6. *Elbaum S., Malishevsky A., G.Rothermel G.* Prioritizing test cases for regression testing // Proceedings of the International Symposium on Software Testing and Analysis. – 2000. – P. 102–112.
7. *Elbaum S., Malishevsky A.G., G.Rothermel G.* Test case prioritization: A family empirical studies // IEEE Transactions on Software Engineering. – 2002. – Vol. 28, N 2. P. 159–182.
8. *Harrold M., Soffa M.* Interprocedural data flow testing // Proceedings of the ACM SIGSOFT third symposium on Software testing, analysis, and verification. – 1989. – P. 158–167.
9. *Agrawal H., Horgan J., Krauser E., London S.* Incremental regression testing // International Conference on Software Maintenance. – 1993. – P. 348–357.
10. *Leung H., White L.* A firewall concept for both control-flow and data-flow in regression integration testing // Proceedings of the Conference on Software Maintenance. – 1992. – P. 262–270.
11. *Chen Y., Rosenblum D., Vo K. TestTube*: A system for selective regression testing // Proceedings of the 16th International Conference on Software Engineering. –1994. – P. 211–222.
12. *Vokolos F., Frankl P.* A regression test selection tool based on textual differencing // Proceedings of the 3rd International Conference on Reliability, Quality & Safety of Software-Intensive Systems. –1997. – P. 3–21.
13. *Frankl P., Rothermel G., Sayre K., Vokolos F.* An empirical comparison of two safe regression test selection techniques. Proceedings of the 2003 International Symposium on Empirical Software Engineering. – 2003. – P. 195–204.
14. *Laski J., Szermer W.* Identification of program modifications and its applications in software maintenance // Proceedings of the Conference on Software Maintenance. – 1992. – P. 282–290.
15. *Rothermel G., Harrold M.* A safe, efficient regression test selection technique // ACM Transactions on Software Engineering and Methodology. – 1997. – Vol. 6, N2. – P. 173–210.
16. *Ball T.* On the limit of control flow analysis for regression test selection // ISSTA: Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis. – 1998. – P. 134–142.
17. *McGregor J., Sykes D.* A Practical Guide to Testing Object-Oriented Software. Addison-Wesley. – 2001. – 416 p.
18. *Kung D., Gao J., Hsia P., Wen F., Toyoshima Y., Chen C.* On regression testing of object oriented programs // Journal of Systems and Software. –1996. – Vol. 32, N1. – P. 21–40.
19. *Jang Y., Munro M., Kwon Y.* An improved method of selecting regression tests for C++ programs // Journal of Software Maintenance: Research and Practice. – 2001. – Vol. 13, N5. – P. 331–350.
20. *Rothermel G., Harrold M.* Selecting regression tests for object-oriented software // International Conference on Software Maintenance. – March 1994. – P. 14–25.
21. *Rothermel G., Harrold M., Dedhia J.* Regression test selection for C++ software // Software Testing, Verification and Reliability. – 2000. – Vol. 10, N2. – P. 77–109.
22. *Harrold M., Jones J., Li T., Liang D., Orso A., Pennings M., Sinha S., Spoon S. A., Gujarathi A.* Regression test selection for Java software // Proceedings of the 16th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, January 2001. – P. 312–326.
23. *Orso A., Shi N., Harrold M.* Scaling regression testing to large software systems // Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering. – 2004. – P. 241–251.
24. *Mansour N., Statieh W.* Regression test selection for C# programs // Advances in Software Engineering. – 2009. – P. 1–10.
25. *Briand L., Labiche Y., Soccar G.* Automating impact analysis and regression test selection based on UML designs Proceedings of the International Conference on Software Maintenance. – 2002. – P. 252–261.
26. *Briand L., Labiche Y., He S.* Automating regression test selection based on UML designs // Information and Software Technology. – 2009. – Vol. 51, N9. – P. 16–30.
27. *Ali A., Nadeem A., Iqbal Z., Usman M.* Regression testing based on UML design models // Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing. – 2007. – P. 85–88.
28. *Farooq Q., Iqbal M., Malik Z., Riebisch M.* A model-based regression testing approach for evolving software systems with flexible tool support. In 17th IEEE International Conference on Engineering of Computer-Based Systems. – March 2010. – P. 41–49.
29. *Naslavsky L., Richardson D.* Using traceability to support model-based regression testing // Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ASE. November 2007. – P. 567–570.
30. *Gorthi, R. Pasala A., Chanduka K., Leong B.* Specification-based approach to select regression test suite to validate changed software // Proceedings of 15th Asia-Pacific Software Engineering Conference. – 2008. – P. 153–160.
31. *Chen Y., Probert R., Sims D.* Specification based regression test selection with risk analysis // Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research. – 2002. – P. 322–323.
32. *Chittimalli P., Harrold M.* Regression test selection on system requirements // Proceedings of the 1st conference on India software engineering conference, August 2008. – P. 87–96.
33. *Orso A., Harrold M., Rosenblum D.* Component metadata for software engineering tasks // Revised Papers from the Second International Workshop on Engineering Distributed Objects, EDO. Springer-Verlag. – 2000. – P. 129–144.
34. *Orso A., Harrold M., Rosenblum D., Rothermel G., Soffa M., and Do H.* Using component metacontent to support the regression testing of component-based software Proceedings of the IEEE International Conference on Software Maintenance, November 2001. – P. 716–725.
35. *Mao C., Lu Y.* Regression testing for component-based software systems by enhancing change information // APSEC: Proceedings of the 12th Asia-Pacific Software Engineering Conference. IEEE Computer Society, December 2005. – P. 611–618.
36. *Mao C., Lu Y., Zhang J.* Regression testing for component-based software via built-in test design // Proceedings of the 2007 ACM symposium on Applied computing. – 2007. – P. 1416–1421.
37. *Sajeev A., Wibowo B.* Regression test selection based on version changes of components // Proceedings of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference, December 2003. – 78 p.
38. *Wu Y., Offutt J.* Maintaining evolving component-based software with UML // Proceedings of 7th European Conference on Software Maintenance and Reengineering, March 2003. – P. 133–142.
39. *Pasala A., Fung Y., Akladios F., Raju A., Gorthi R.* Selection of regression test suite to validate software applications upon deployment of upgrades // 19th Australian Conference on Software Engineering, March 2008. – P. 130–138.
40. *Zheng J., Robinson B., Williams L., Smiley K.* Applying regression test selection for COTS based applications // ICSE: Proceedings of the 28th international conference on Software engineering, May 2006. – P. 512–522.
41. *Zhao J., Xie T., Li N.* Towards regression test selection for AspectJ programs // Proceedings of the 2nd workshop on Testing aspect-oriented programs. – 2006. – P. 21–26.
42. *Xu G., Rountev A.* Regression test selection for AspectJ software // Proceedings of the 29th international conference on Software Engineering, December 2007. – P. 65–74.

43. *Haftman F., Kossmann D., Lo E.* A framework for efficient regression tests on database applications // The VLDB Journal. – 2007. – Vol. 10, N 2. – P. 145–164.
44. *Haraty R., Mansour N., Daou B.* Advanced Topics in Database Research. Idea Group. – 2004. – Vol. 3. – P. 141–165.
45. *Willmor D. and Embury S.* A safe regression test selection technique for database-driven applications // Proceedings of the 21st IEEE International Conference on Software Maintenance.– 2005. – P. 421–430.
46. *Xu L., Xu B, Chen Z., Jiang J., Chen H.* Regression testing for web applications based on slicing // Proceedings of the 27th Annual International Computer Software and Applications Conference. – 2003. – P. 652–656.
47. *Tarhini A., Fouchal H., Mansour N.* Regression testing web services-based applications // AICCSA Proceedings of the IEEE International Conference on Computer Systems and Applications. – 2006. – P. 163–170.
48. *Ruth M., Tu S.* A safe regression test selection technique for web services // Proceedings of the Second International Conference on Internet and Web Applications and Services. – 2007. – P. 47.
49. *Lin F., Ruth M., Tu S*. Applying safe regression test selection techniques to Java web services // International Conference on Next Generation Web Services Practices. – 2006. – P. 133–142.
50. *Rothermel G., Untch R., Chu C., Harrold M.J.* Test case prioritization: An empirical study // Proceedings of International Conference Software Maintenance, August 1999. – P. 179–188.
51. *Harrold M. J., Larsen L., Lloyd J., Nedven D., Page M., Rothermel G., Manvinder S., Smith M.* Aristotle: A System for Development of Program Analysis Based Tools // Proceedings of the 33rd annual on Southeast regional conference. – 1995. – P. 110–119.
52. *Srivastava A. Thiagarajan J.* Effectively prioritizing tests in development environment // Proceedings of the International Symposium on Software Testing and Analysis. – 2002. – P. 97–106.
53. *Belli F., Eminov M., Gokco N*. Coverage-Oriented, Prioritized Testing-A Fuzzy Clustering Approach and Case Stud // Proceedings of the Third Latin-American conference on Dependable Computing. – 2007. – P. 95–110.
54. *Do H., Mirarab S., Tahvildari L., Rothermel G.* An Empirical Study of the effect of time constraints on the cost benefits of regression testing // Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. – 2008. – P. 71–82.
55. *Jiang B., Zhang Z., Chan W.K., Tse T.H.,* Adaptive Random test case prioritization // Proceedings of International Conference on Automated Software Engineering. – 2009. – P. 233–243.
56. *Bryce R.C., Sampath S., Memon A.M.* Developing a Single Model and Test Prioritization Strategies for Event Driven Software // Transactions on Software Engineering. – 2010. – P. 48–63.
57. *Wong W.E., Horgan J.R., London S., Aggarwal A*. A study of effective regression testing in practice // Proceedings of the Eighth International Symposium Software Reliability Engineering, November 1997. – P. 230–238.
58. *Korel B., Tahat L., Harman M.,* Test Prioritization Using System Models // The Proceedings of 21st International Conference on Software Maintenance. – 2005. – P. 247–256.
59. *Korel B., Koutsogiannakis G., Talat L.H.* Modelbased test suite prioritization Heuristic Methods and Their Evaluation // Proceedings of 3rd workshop on Advances in model based testing. – 2007. – P. 34–43.
60. *Filho R.S.S., Budnik C.J., Hasling W.M., Kenna M.M., Subramanyam R.* Supporting concern based regression testing and prioritization in a model driven environment // Proceedings of 34th Annual Computer software and Applications conference Workshops. – 2010. – P. 323–328.
61. *Srikanth H., Williams L., Osborne J.* System Test Case Prioritization of New and Regressionb Test Cases // Proceedings of International Symposium on Empirical Software Engineering. – 2005. – P. 64–73.
62. *Krishnamoorthi R., Mary S.A.* Incorporating varying requirement priorities and costs in test case prioritization for new and regression testing // Proceedings of International Conference on Computing, Communication and Networking. – 2008. – P. 1–9.
63. *Krishnamoorthi R., Mart S.A.* Factor oriented requirement coverage based system test case prioritization of new and regression test cases // Journal of information and software technology. – 2009. – Vol. 51. – P. 799–808.
64. *Kim J.M., Porter A*. A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environment // Proceedings of the 24th International Conference Software Engineering, May 2002. – P. 119–129.
65. *Fazlalizadeh Y., Khalilian A., Azgomi H.A., Parsa S.* Incorporating historical test case performance data and resource constraints into test case prioritization // Lecture notes in Computer Science, Springer. – 2009. – P. 43–57.
66. *Walcott K.R., Soffa M.L., Kapfhammer G.M., Roos R.S.* Time aware test suite Prioritization // Proceedings of International Symposium on software Testing and Analysis, July 2006. – P. 1–12.
67. *Conrad A.P., Roos R. S.* Empirically Studying the role of selection operators during search based test suite prioritization // Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference, Portland, Oregon. – 2010. – P. 1373–1380
68. *Smith A.M., Kapfhammer G.M.* An empirical study of incorporating cost into test suite reduction and prioritization // Proceedings of ACM Symposium on Applied Computing. – 2009. – P. 461–467.
69. *Yoo S., Harman M.* Pareto efficient multi-objective test case selection // Proceedings of the 2007 International Symposium on Software Testing and Analysis. – 2007. – P. 140–150.
70. *The* Mathworks, Inc. MATLAB. Website, April 2011. http://www.mathworks.com.
71. *Jennings N., Wooldridge M.* Agent-Oriented Software Engineering // Artificial intelligence. – 2000. – Vol. 117. – P. 277–296.