УДК 004.415.53

ПРИОРИТЕЗАЦИЯ ТЕСТОВ КАК МЕТОД БЫСТРОГО ВЫЯВЛЕНИЯ СЕРЬЕЗНЫХ ОШИБОК

А.Г. МАЛЫШЕВСКИЙ

Рассмотрено использование информации о стоимости тестов и серьезности ошибок в процессе приоритезации в регрессивном тестировании. Описаны способы оценки стоимости тестов и серьезности ошибок. Рассмотрена серьезность ошибок с точки зрения надежности программы. Исследованы три метода внедрения информации о стоимости и критичности тестов в процесс приоритезации. Исследовано влияние шкалы стоимости тестов и серьезности ошибок на эффективность методов приоритезации.

ВВЕДЕНИЕ

Одним из методов проверки, удовлетворяет ли программа заданным требованиям и ее спецификации, является тестирование (выполнение программы и проверка ее поведения на соответствие спецификациям). Каждый тест из набора тестов T, использующегося в тестировании, состоит из множества входных данных (сценариев тестирования), состояния системы до получения этих входных данных и ожидаемое состояние системы вместе с выходными данными. Обычно набор тестов создается, исходя из некоторого множества правил, называемого критерием адекватности. Данный критерий выражает условия, которым должен удовлетворять набор тестов [1]. На стадии сопровождения программы многократно используется регрессивное тестирование, которое показывает, что внесенные в программу модификации не только изменили программу в соответствии с новыми спецификациями и исправили найденные ошибки, но и не внесли новых ошибок [1].

В регрессивном тестировании растет набор тестов, увеличивая стоимость и продолжительность тестирования. Например, одна из систем ПО из 20 000 строк кода требует семь недель для тестирования при использовании всех тестов в наборе. Во многих случаях в процессе регрессивного тестирования можно использовать только подмножество набора тестов для проверки модифицированной программы. Но иногда бывает сложно использовать или совсем не допускается использование неполного набора тестов, например, для программ, надежность которых является критичной (авионика или управление медицинским оборудованием). В данном случае для уменьшения стоимости регрессивного тестирования может быть применен иной подход: тесты упорядочиваются (приоритезируются) для регрессивного тестирования таким образом, чтобы более важные из них выполнялись в первую очередь.

Вопросам приоритезации уделено много внимания [2–22]. В методах приоритезации тесты сортируют таким образом, чтобы эффективнее достичь заданной цели, например, наиболее быстрого покрытия операторов программного кода, функций программы в порядке частоты их использования или подсистем в порядке частоты их сбоев в прошлом. Возможная цель приоритезации — увеличение скорости выявления ошибок набором тестов в процессе тестирования. Возросшая скорость выявления ошибок может обеспечить более раннюю обратную связь с регрессивно тестируемой системой и позволить разработчикам начать поиск местонахождения ошибок, а также их исправление раньше, чем это было бы возможно в ином случае. Такая обратная связь обеспечивает выявление ранних признаков того, что заданные цели еще не достигнуты, и позволяет принимать стратегические решения о сроках релиза на ранних этапах. Повышенная скорость обнаружения ошибок увеличивает вероятность того, что в случае преждевременного прекращения процесса тестирования тесты, обеспечивающие наибольшую способность выявлять ошибки в сроки, выделенные на тестирование, уже были выполнены.

В работах [4, 5, 7, 10, 13, 20] была представлена метрика $APFD$ (weighted average of the percentage of faults detected — взвешенное среднее процента выявленных ошибок) для измерения скорости выявления ошибок во время выполнения набора тестов в заданном порядке. В этих работах представлены несколько методов приоритезации, целью которых являлось увеличения скорости выявления ошибок, а также была эмпирически оценена их эффективность. Результаты показали, что некоторые методы приоритезации (например, приоритезация по дополнительному покрытию операторов) могут ускорить выявление ошибок.

На практике, как стоимость тестов, так и серьезность ошибок могут значительно отличаться между собой. Поэтому, в последующих работах [6, 9, 21, 22] была рассмотрена приоритезация учитывающая стоимость тестов и серьезность ошибок. Также была разработана метрика $APFD_C$ учитывающая их, и было проиллюстрировано применение приоритезации.

Цель работы — исследование альтернативного подхода и внедрению информации о серьезности потенциальных ошибок в процесс приоритезации используя реальную информацию о стоимости тестов и серьезности ошибок.

В данной работе, мы расширим исследования представленные в [6, 9, 21, 22] следующим образом. Мы более детально рассмотрим применение приоритезации учитывающей стоимости тестов и серьезности ошибок, рассмотрим серьезность ошибок с точки зрения их влияния на надежность программы (в предшествующих исследованиях, мы рассматривали серьезность как последствия индивидуальных сбоев программы вызванных данной ошибкой), рассмотрим альтернативный подход к внедрению информации о серьезности потенциальных ошибок в процесс приоритезации, рассмотрим большее, нежели в предыдущих работах количество методов приоритезации, используем реальную, вместо искусственно сгенерированной, информацию о стоимости тестов и серьезности ошибок, рассмотрим их различные шкалы, а также проведем расширенные исследования, в которых используем большее количество программ-обьектов исследования.

Научная новизна данной работы состоит в разработке и изучении нового подхода внедрения стоимости тестов и серьезности ошибок в процесс приоритезации.

МЕТРИКИ *APFD* И $APFD_C$

Для анализа эффективности приоритезации необходимо оценить ее количественно.

В ранних исследованиях [4, 5, 7, 10, 13, 20] использовалась метрика $APFD$, оценивающая скорость выявления ошибок набором тестов в интервале от 0 до 100. Чем больше значение метрики, тем быстрее выявляются ошибки. Метрика $APFD$ проиллюстрирована на рис. 1. Кривая на графике отмечает выявленные в процессе тестирования ошибки. Численное значение метрики — выраженная в процентах площадь под кривой (отмечена серым цветом) относительно площади всего прямоугольника. Рис. 1,*а* содержит таблицу показывающую какие ошибки (от 1 до 10) выявляет каждый тест (A, B, C, D или E). Четыре части рисунка (б, в, г, д) графически отображают метрику для четырех различных порядков тестов.

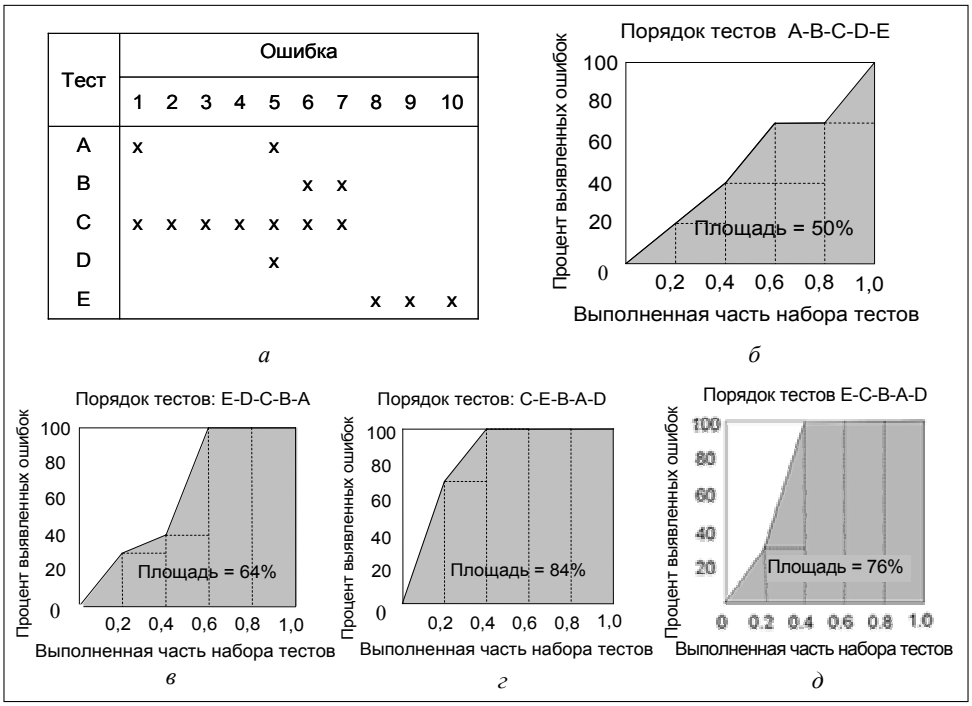
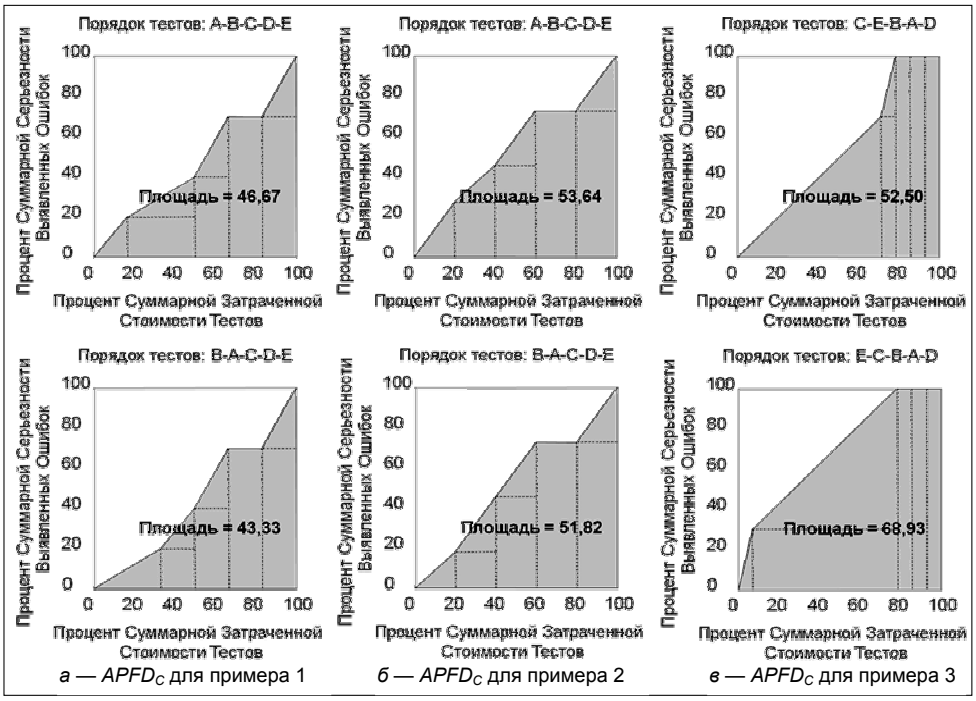


Рис. 1. Примеры, иллюстрирующие метрику APFD: *а* — тесты и выявленные ими ошибки; *б* — APFD для приоритезированного набора тестов T1; *в* — для T2; *г* — для T3; *д* —для T4

Однако данная метрика основывается на двух допущениях: 1) все ошибки идентичны по серьезности; 2) все тесты идентичны по стоимости. Эти допущения выражаются в том, что данная метрика просто определяет процент выявленных ошибок для выполненной части набора тестов. Поэтому в более поздних работах [6, 9, 21, 22], где учитывалась как стоимость тестов так и серьезность ошибок, была разработана и использована метрика $APFD_C$. Эта метрика используется и в данной работе. В метрике $APFD_C$, на графике, на горизонтальной оси отображается «Процент суммарной затраченной стоимости тестов». Здесь, каждый тест в наборе представлен интервалом вдоль горизонтальной оси, чья длина пропорциональна доли стои-

мости данного теста в суммарной стоимости тестов в наборе. На вертикальной оси графика, отображается «Процент суммарной серьезности выявленных ошибок». Теперь, каждая ошибка выявленная набором тестов представлена интервалом вдоль вертикальной оси, чья высота пропорциональна доли ее серьезности в общей сумме серьезности ошибок. В свете этой новой интерпретации, на графиках, вклад теста взвешивается в горизонтальном направлении по его стоимости, и вдоль вертикального направления по суммарной серьезности выявленных им ошибок. В таких графиках, кривая ограничивает большую площадь для порядка тестов который демонстрирует больше «единиц серьезности ошибок выявленных на единицу стоимости теста». Эта площадь и составляет метрику $APFD_C$.



а — $APFD_C$ для примера 1 *б* — $APFD_C$ для примера 2 *в* — $APFD_C$ для примера 3

Рис. 2. Примеры, иллюстрирующие метрику $APFD_C$

Для иллюстрации метрики $APFD_C$ графически, рис. 2 показывает графики для трех примеров. Информация о выявлении ошибок тестами взята из рис. 1,*а*.

Пример 1. Рассмотрим сценарий, показанный на рис. 1. Допустим, что стоимость теста B в два раза превосходит стоимость теста A, требующего два часа машинного времени, тогда как тест A — один час. С точки зрения ошибок, выявленных за час, порядок тестов A–B–C–D–E предпочтительней порядка B–A–C–D–E, который выявляет ошибки быстрее. Пара графиков расположенная слева (рис. 1,*а*) соответствует примеру 1: верхний график представляет метрику $APFD_C$ для порядка тестов A–B–C–D–E, а нижний график — метрику $APFD_C$ для порядка B–A–C–D–E. Метрика $APFD_C$ дает предпочтение порядку A–B–C–D–E, который быстрее выявляет ошибки.

Пример 2. Работая опять со сценарием, показанным на рис. 1, предположим, что все пять тестов имеют равную стоимость и что ошибки 2…10 имеют значение серьезности, равное k, тогда как ошибка 1 имеет серьез-

ность $2k$. В данном случае тест A выявляет одну более серьезную ошибку и одну менее серьезную, тогда как тест B — только две менее серьезные ошибки. С точки зрения скорости выявления суммарной серьезности ошибок порядок тестов A–B–C–D–E предпочтительней порядка B–A–C–D–E. Пара графиков на рис. 1,*б,* соответствующая примеру 2, показывает как метрика $APFD_C$ дает более высокую оценку порядку тестов, который раньше выявляет более серьезную ошибку (A–B–C–D–E) при допущении, что ошибкам 2–10 присвоено значение серьезности k и ошибке 1 присвоено значение серьезности $2k$.

Пример 3. Предположим, что все десять ошибок равнозначны по серьезности и, что каждый из тестов A, B, D и E требует один час для выполнения, а тест C — десять часов. Однако с точки зрения ошибок, выявленных за единицу времени, второй порядок (E–C–B–A–D) предпочтительнее: он выявляет три ошибки в течение первого часа и остается лучшим, чем первый порядок, до конца выполнения второго теста. Аналогичный пример может быть приведен для использования ошибки с неравной серьезностью при равной стоимости тестов. Пара графиков на рис. 1,*в*, соответствующие примеру 3, показывает как метрика различает порядки тестов, где тест C имеет высокую стоимость: для порядка E–C–B–A–D, метрика присваивает ему большее значение, чем порядку C–E–B–A–D.

Пусть T будет набором содержащим n тестов со стоимостями $t_1, t_2, \ldots, t_n$; F — множеством m ошибок выявленных набором тестов T; и $f_1, f_2, \ldots, f_m$ — значениями серьезности этих ошибок. Пусть TF_i будет первый тестом в порядке T' набора T, который выявляет ошибку i. Тогда формула для метрики $APFD_C$ имеет следующий вид:

$$APFD_C = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2}t_{TF_i}\right)\right)}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i}.$$

СТОИМОСТЬ ТЕСТОВ

Стоимость теста — это количество ресурсов (например, денег или времени) затраченных на его выполнение (не включает в себя затраты на его разработку). Существует два вопроса связанных со стоимостью тестов: первый — измерение или оценка стоимости с целью вычисления значения метрики $APFD_C$ для порядка тестов и второй — оценка стоимости для использования в приоритезации тестов. Стоимость теста связана с ресурсами затраченными на выполнение данного теста и проверку результатов. Стоимость теста может измерять фактическое время затраченное на тестирование программы заданным тестом. Кроме того, стоимость может включать денежные затраты на выполнение теста и проверку результатов, включая стоимость оборудования, зарплату, стоимость материалов требующихся для тестирования, потери дохода связанные с задержкой выпуска ПО, потери связанные со срывом сроков релиза и т.д.

Определить стоимость тестов после завершения тестирования несложно, что и необходимо сделать для вычисления метрики $APFD_C$. В данном случае, нам необходимо обратить внимание на то, какие ресурсы были затрачены на каждый тест. Гораздо сложнее оценить стоимость тестов перед началом тестирования, что необходимо для использования стоимости в процессе приоритезации. В данном случае, нам необходимо предугадать стоимость тестов. Одним способом является анализ теста и программного кода выполненного тестом. Другим способом, который мы используем в данной работе является использование собранных данных о стоимости тестов на предыдущих сессиях тестирования (что представляется возможным при регрессивном тестировании). Здесь, мы полагаем, что стоимость тестов не меняется значительно от одной версии программы к другой.

В наших экспериментах для регрессивного тестирования требовалось лишь время выполнения каждого теста. Для сбора данных о времени их выполнения, все тесты, один за другим, были выполнены, и время выполнения и валидации результатов было измерено. То есть, в данной работе, стоимость теста — это время его выполнения в секундах.

СЕРЬЕЗНОСТЬ ОШИБОК

Стоимость ошибки — это затраты или потери, выраженные в неких единицах (например деньгах), имеющие место, если ошибка осталась в релизе. Как и со стоимостью тестов, существует два вопроса связанных с серьезностью ошибок: первый — их измерение или оценка для вычисления метрики $APFD_C$ для порядка тестов и второй — их оценка для использования информации при приоритезации тестов. Серьезность ошибки связана с затратами произошедшими, если ошибка осталась в программе после ее релиза. Возможны различные подходы для измерения такой серьезности, например:

1. Можно измерить серьезность ошибки как сумму денег потерянных в результате сбоя вызванного данной ошибкой (с учетом вероятности такого сбоя). Такой подход может применяться в ПО, где индивидуальные сбои могут привести к катастрофическим последствиям (например, судебные иски, человеческие жертвы, потеря оборудования).

2. Можно оценить последствия ошибки на надежность ПО. Данный подход применяется в ПО, где ошибки могут вызвать лишь неудобство для пользователей, и маловероятно, что сбой будет иметь серьезные последствия (например, текстовый редактор). В данном случае, основным последствием является снижение надежности ПО, которое может привести к потере клиентов.

В предыдущих работах [6, 9, 21, 22] использовался первый подход. В данной работе, мы рассмотрели и применили второй подход.

Подобно ситуации со стоимостью тестов, нас интересует как оценка серьезности ошибок до их обнаружения, так и оценка их серьезности после обнаружения. Когда тестирование закончено и уже есть информация об ошибках, можно оценить их серьезность для использования в метрике $APFD_C$ (хотя это не так просто как со стоимостью тестов). С другой стороны, перед началом тестирования необходимо оценить серьезность потенциальных ошибок для использования данной информации в процессе приори-

тезации, а это намного сложнее, чем с оценкой стоимости тестов. Поэтому, необходимо связать тесты с серьезностью ошибок, которые они выявляют.

Если иметь информацию о том, какие ошибки выявляет каждый тест, и знать их серьезность, было бы несложно связать тесты с серьезностью ошибок. Но на практике, эта информация недоступна до завершения тестирования. Существуют два подхода для такой оценки.

1. Оценка критичности модулей. Здесь необходимо связать серьезность ошибки с критичностью модуля (или любого другого компонента программного кода), в котором данная ошибка может содержаться.

2. Оценка критичности тестов. Здесь необходимо связать тесты с серьезностью ошибок, которые они могут выявить.

Оценив критичность модулей или тестов, необходимо включить информацию о серьезности ошибок в процесс приоритезации. В данной работе, в исследованиях мы применим второй подход. Критичность тестов используется как метрика для оценивания серьезности ошибок, которые может выявить каждый тест. Так как на практике мы не можем знать об ошибках наперед, необходимо оценить критичность тестов не используя информацию об ошибках, которые данный тест может выявить. Полагаем, что серьезность ошибки определяется ее влиянием на надежность системы. Чем серьезнее ошибка, тем чаще происходят связанные с ней сбои. В данной работе мы не дифференцируем сбои, а учитываем лишь их частоту.

Для вычисления серьезности ошибок, нам необходимы понятия операции и операционного профиля.

Операции и операционный профиль

Для понимания концепции *операционный профиль*, нам необходимо определить, что такое операция. Операцией является внешне-инициированное действие выполненное системой [23]. Операция может быть сравнена с функцией, внешне-инициированным действием, с точки зрения пользователей. Примерами операций могут быть команды, транзакции и обработка внешних событий.

Операционный профиль (operational profile) является множеством операций и вероятностью их использования [23]. Другими словами, дана система S, операционный профиль для S — список действий, которые пользователи могут делать с S, и частоту этих действий.

Рассмотрим пример текстового редактора. В этом примере, операция «изменить шрифт» может выполняться 20% времени, операция «вырезать текст» — 10%, «ввести текст» — 50%, «копировать текст» — 10% и «вставить текст» — 10%.

Операционный профиль часто используется при разработке систем ПО, включающим дизайн, разработку, тестирование и использование. Например, одним применением операционного профиля может быть тестирование операций пропорционально частоте их использования. Например, при тестировании текстового редактора приведенного в нашем примере, нам необходимо потратить половину усилий тестируя операцию «вставка текста».

В данной работе, мы использовали операционный профиль для оценки серьезности ошибок и критичности тестов. Для объектов исследования, сначала было определено множество операций применимых к программам, а затем операции были связаны с тестами. Это позволило создать матрицу

операций для программ, паруя операции с тестами. Матрица операций используется для определения серьезности ошибок и критичности тестов описанным позднее методом.

Для использования серьезности ошибок в алгоритмах приоритезации необходимо ввести понятие критичности тестов.

Критичность тестов и ее вычисление

Была определена критичность теста как его важность для пользователей программы. Критичность тестов может быть ассоциирована с частотой использования различных операций в системе. Тест выполняющий чаще используемые операции в программе более критичен, чем тест выполняющий реже используемые операции. Мы вычислили критичность каждого теста как сумму вероятностей использования различных операций выполненных тестом. То есть, критичность теста $t_{\text{crit}} = \sum_{i=1}^{n} P_i$, где n — количество операций выполненных тестом и P_i — вероятность использования i-й операции.

Вычисление серьезности ошибок

В данной работе мы используем операционный профиль для оценки серьезности ошибок. Полагаем, что сбои в чаще используемых операциях ведут к более частым сбоям при нормальном использовании программы, и таким образом могут создавать более значительные проблемы пользователям. Поэтому, таким сбоям присваивается более высокая серьезность нежели сбоям связанным с редко используемыми операциями.

Для определения серьезности ошибок для наших объектов исследования, используется связь между матрицей ошибок (какие тесты выявляют какие ошибки) и матрице операций (какие тесты выполняют какие операции). Сначала определяем ассоциацию между ошибками и операциями. Эта связь определяется матрицей ошибок и матрицей операций.

Вернемся к примеру текстового редактора. Допустим в программе присутствуют три ошибки. Пусть ошибки 1 и 2 затрагивают операции «изменить шрифт», «вырезать текст», «ввести текст». Ошибка 3 затрагивает операции «копировать текст» и «вставить текст». В этом же примере, тесты 1, 2 и 3 выполняют операции «изменить шрифт», «вырезать текст» и «ввести текст», а тесты $4-8$ выполняют операции «копировать текст» и «вставить текст». Кроме этого, ошибка 1 выявляется тестами 1, 2, 3, ошибка 2 выявляется тестом 1, а ошибка 3 выявляется тестами $4-8$.

Таким образом, серьезностью ошибки может быть сумма вероятности использования операций затронутых ошибкой. Например, серьезность ошибки 1 — это сумма вероятности операций затронутых ошибкой 1, и равняется $0{,}20 + 0{,}10 + 0{,}50 = 0{,}80$ (из описанного выше примера). Серьезность ошибки 2 равняется $0{,}80$, а серьезность ошибки 3 равняется $0{,}20$.

Такой подход дает ошибкам 1 и 2 равную серьезность. Ошибка 1 выявляется тестами 1, 2 и 3, но, с другой стороны, ошибка 2 выявляется только тестом 1. Если тесты соответствуют операционному профилю, то операции затронутые большим количеством тестов должны давать больший вклад при подсчете серьезности ошибки, которую они затрагивают.

Для разрешения данной проблемы, учитывается количество тестов ассоциированных с каждой операцией. Если на операцию влияют две ошибки,

f_1 и f_2, и если f_1 выявляется большим количеством тестов нежели f_2, придается большая серьезность ошибке f_1 нежели ошибке f_2.

Одним из путей для присвоения более высокой серьезности ошибке 1, чем ошибке 2 — это использование количества тестов как веса при вычислении серьезности ошибок. Пусть T_i — список тестов ассоциированных с операцией *i*. Пусть Op — множество m операций связанных с ошибкой X, а P_i — вероятность ассоциированная с операцией *i*. Серьезность ошибки X может быть определена как $\sum_{i=1}^{m} P_i \times \|T_i\|$.

По этому методу, серьезность ошибок 1, 2 и 3 в наших примерах следующие: серьезность ошибки 1 равна $3*0{,}20+3*0{,}10+3*0{,}50=2{,}40$; серьезность ошибки 2 равна $0{,}20+0{,}10+0{,}50=0{,}80$; серьезность ошибки 3 равна $5*0{,}10+5*0{,}10=1{,}0$.

Однако этот метод делает ошибку 3 более серьезной нежели ошибку 2, а это неправильно, так как ошибка 2, в отличие от ошибки 3, может влиять на наиболее часто используемую операцию. Это противоречит нашему утверждению, что ошибки встречающиеся в наиболее часто используемых операциях более серьезны, чем ошибки которые встречаются в менее часто используемых операциях. Поэтому, вводится некое значение барьера выше которого будет учитываться количество тестов и ниже которого — нет. Пусть P_H — вероятность использования наиболее часто используемой операции на которую влияет данная ошибка, а P_L — вероятность использования наименее часто используемой операции на которую влияет данная ошибка. Значение барьера для TH дано формулой: $TH=(P_H+P_L)/2$.

Таким образом, значение барьера для примера с текстовым редактором будет следующее: $TH=(0{,}50+0{,}10)/2=0{,}30$. Поэтому, серьезность ошибок основанная на данном значении барьера будет: серьезность ошибки 1 равна $0{,}20+0{,}10+3*0{,}50=1{,}80$; серьезность ошибки 2 равна $0{,}20+0{,}10+$ $+0{,}50=0{,}80$; серьезность ошибки 3 равна $0{,}10+0{,}10=0{,}20$.

ЭМПИРИЧЕСКИЕ ИССЛЕДОВАНИЯ

Было проведено исследование, цель которого изучить, как различные распределения и шкалы стоимости тестов и серьезности ошибок могут влиять на скорость их выявления с точки зрения метрики $APFD_C$. Понятие критичности теста применялось для оценки серьезности ошибок в приоритезации. Рассмотрено влияние различных распределений и шкал стоимости тестов, серьезности ошибок и их комбинаций на относительную эффективность методов приоритезации.

Объекты исследования

Как объекты данного исследования были использованы следующие программы:

- **empire** — сетевая игра состоящая из 63 014 – 64 396 строк кода, 10 версий, 1985 тестов и по 10 регрессивных ошибок во всех версиях, кроме первой;

- **grep** — программа поиска текста состоящая из 10 929 – 13 359 строк кода, 5 версий, 810 тестов и по от одной до четырех регрессивных ошибок во всех версиях, кроме первой;
- **flex** — генератор лексических анализаторов состоящий из 11 783 – 14 171 строк кода, 5 версий, 568 тестов и по от одной до восьми регрессивных ошибок во всех версиях, кроме первой;
- **make** — программа применяющаяся для построения ПО состоящая из 18 665 – 25 465 строк кода, 5 версий, 1044 тестов и по от трех до пяти регрессивных ошибок во всех версиях, кроме первой;
- **sed** — потоковый редактор состоящий из 8 063 – 11 911 строк кода, 3 версии, 1294 тестов и по от четырех до пяти регрессивных ошибок во всех версиях, кроме первой;
- **xearth** — программа отображающая картинку земного шара, состоящая из 13 165 – 24 179 строк кода, трех версий, 540 тестов и по от трех до четырех регрессивных ошибок во всех версиях, кроме первой.

Используя документацию для программы **empire**, были определены 182 операции, обычно используя прямую ассоциацию с командами. Так как каждый тест выполняет последовательность команд, было легко ассоциировать тест с затронутыми им операциями. Для других программ, тесты были сгенерированы исходя из языка спецификации тестов (test specification language (TSL)). Для каждой программы, была создана спецификация тестов базируясь на документации. Далее эти спецификации были преобразованы в тестовые фреймы (test frames) из которых были сгенерированы тесты. Операции были созданы для каждого выбора в файле спецификации тестов; затем некоторые операции с общими характеристиками были объединены между собой. Далее, используя информацию о том как были сгенерированы тесты, они были увязаны с операциями.

Для получения операционного профиля для программы **empire**, было задействовано несколько добровольцев, которые на протяжении некоторого времени играли в эту игру. Для других программ, операционный профиль был сгенерирован случайным образом.

В области разработки программного обеспечения очень сложно найти репрезентативную выборку программ, так как существует слишком большие между ними различия, не только в их размерах, используемых языках программирования и стилях программирования, но и в классах решаемых задач. Для репрезентативного набора необходимо иметь доступ ко всем программам и случайным образом выбрать их подмножество, что не является возможным, поэтому был использованы программы, которые могут быть репрезентативными для своего класса (программы размером 8,000 – 65,000 строк, написанные на языке C и выполняющие роль утилит Unix и сетевой игры), но не являются репрезентативными для всех программ. Для получения более общих результатов необходимо собрать большее количество программ различных классов. Поэтому задачей данных исследований является не нахождение обобщающихся зависимостей, а эксплораторное исследование применяемых подходов для нахождения тенденций и иллюстрации их влияния на процесс приоритезации.

Методы приоритезации

В исследовании были использованы следующие методы приоритезации:

- **fn-nofb** — приоритезация по общему покрытию функций. Вставляя в программу дополнительный код, можно определить для каждого теста, какие функции им покрыты. Эти тесты можно приоритезировать в соответствии с количеством покрытых ими функций, сортируя тесты в порядке убывания данного общего покрытия.
- **fn-fb** — приоритезация по дополнительному покрытию функций. Приоритезация по общему покрытию функций планирует тесты в порядке достижения совокупного покрытия функций. Однако, выполнив тест и покрыв некоторые функции, целесообразно при последующем тестировании покрыть еще не покрытые функции. Приоритезация по дополнительному покрытию функций выбирает тест, достигший наибольшего покрытия еще не покрытых функций, повторяя этот процесс до тех пор, пока ни один из оставшихся тестов не сможет увеличить покрытие функций. Когда это происходит, данный алгоритм применяется рекурсивно к оставшимся тестам.
- **diff-fn-nofb** — приоритезация по общему покрытию измененных функций. Этот метод похож на **fn-nofb**, но рассматривает только новые или измененные функции. При равенстве покрытых измененных функций, учитывается количество всех покрытых функций.
- **diff-fn-fb** — приоритезация по дополнительному покрытию измененных функций подобна приоритезации по дополнительному покрытию функций, за исключением того, что учитываются только измененные или новые функции. При равенстве покрытых измененных функций, учитывается количество всех покрытых функций.
- **random** упорядочивает тесты случайным образом.

Для того, чтобы учесть стоимость и критичность тестов в методах приоритезации, было найдено отношение критичности теста к его стоимости. В данном исследовании было рассмотрено три метода внедрения этого отношения в методы приоритезации:

- **Af** — тесты приоритезируются в соответствии с покрытием функций (с или без учета модификации программного кода). Отношение критичности теста к его стоимости имеет второстепенное значение и учитывается при равнозначности покрытия.
- **Mult** — тесты приоритезируются в соответствии с произведением покрытия функций (с или без учета модификаций программного кода) и отношения критичности теста к его стоимости. В данном случае вклад покрытия и стоимости с критичностью равнозначны.
- **Rf** — тесты приоритезируются в соответствии с отношением критичности теста к его стоимости, а покрытие функций (с или без учета модификаций программного кода) имеет второстепенное значение и учитывается при равнозначном отношении критичности к стоимости.

Шкалы стоимости тестов

В исследовании были рассмотрены четыре шкалы стоимости тестов.

- **Unit** — данная шкала не различает между стоимостью тестов (в данном случае стоимость теста не учитывается).

- **Orig** — данная шкала присваивает каждому тесту его стоимость (настоящую либо искусственно сгенерированную) полученную описанными ранее методами.
- **Buck-lin** — в данной шкале, значения стоимости тестов (из шкалы **orig**) отсортированы и равномерно распределены в ячейки (стоимость тестов соответствует номеру ячейки начиная с единицы).
- **Buck-exp** — данная шкала подобна **buck-lin**, кроме того, что стоимость теста не номер ячейки n, а $b^n - 1$, где $b = 2$ и n — номер ячейки (данная шкала — альтернатива предыдущей, которая увеличивает значимость наиболее дорогих тестов).

Шкалы критичности тестов и серьезности ошибок

В исследовании были рассмотрены четыре шкалы критичности тестов и серьезности ошибок.

- **Unit** — шкала не различает между критичностью тестов и серьезностью ошибок (в данном случае критичность теста и серьезность ошибок не учитывается).
- **Orig** — данная шкала присваивает каждому тесту его критичность и каждой ошибке ее серьезность полученные описанными ранее методами.
- **Buck-lin** — в данной шкале, значения критичности тестов и серьезности ошибок (из шкалы **orig**) отсортированы и равномерно распределены в ячейки. Критичность тестов и серьезность ошибок вычисляется по номеру ячейки.
- **Buck-exp** — данная шкала подобна **buck-lin**, только вместо номера ячейки n, используется $b^n - 1$, где $b = 2$, а n — номер ячейки. Данная шкала — альтернатива предыдущей, которая увеличивает значимость наиболее серьезных ошибок и соответствующих им тестов.

РЕЗУЛЬТАТЫ

Проведенное исследование состояло из трех частей. В первой части было рассмотрено влияние трех методов внедрения информации о стоимости и критичности тестов в процесс приоритезации. Во второй части было рассмотрено как влияет выбор шкалы стоимости тестов и серьезности ошибок на относительную эффективность методов приоритезации. В третьей части — рассмотрено для каждого метода приоритезации относительное влияние всех 16 комбинаций шкал на скорость выявления ошибок.

Исследование 1. В данном исследовании рассматриваются три метода комбинации информации (**af, mult** и **rf**) о стоимости и критичности тестов с информацией о покрытии элементов программного кода.

В данном исследовании были получены данные используя различные распределения (шкалы) стоимости тестов и серьезности ошибок (**orig**/**orig, buck-lin**/**buck-lin, buck-exp**/**buck-exp**) для каждого из пяти методов приоритезации и трех видов комбинации информации. Был проведен статистический анализ Anova. Результаты представлены в виде диаграмм размаха (рис. 3–6).

Как видно из графиков (рис. 3–6) для методов **fn-nofb** и **diff-fn-nofb**, в трех рассмотренных шкалах стоимости тестов и серьезности ошибок, **af** имел наилучшие результаты; а для методов **fn-fb** и **diff-fn-nofb**, **af** и **mult**

были значительно лучше чем **rf**. То есть, для методов без обратной связи (по общему покрытию функций), комбинация **af** была самая эффективная, а для методов с обратной связью (по дополнительному покрытию функций), как **af** так и **mult** являются эффективными. Анализ Anova показал, что не все результаты являются статистически значимыми (для $\alpha = 0{,}05$). Значимыми являются: метод **fn-nofb** со шкалой **exp/exp** ($p = 0{,}0234$), метод **fn-fb** со шкалами **orig/orig**, **buck-lin/buck-lin** и **buck-exp/buck-exp** ($p = 0$), а также метод **diff-fn-fb** со шкалами **orig/orig** ($p = 2{,}2622 * 10^{-7}$), **buck-lin/buck-lin** ($p = 1{,}926310^{-11}$) и **buck-exp/buck-exp** ($p = 3{,}4632 * 10^{-4}$).

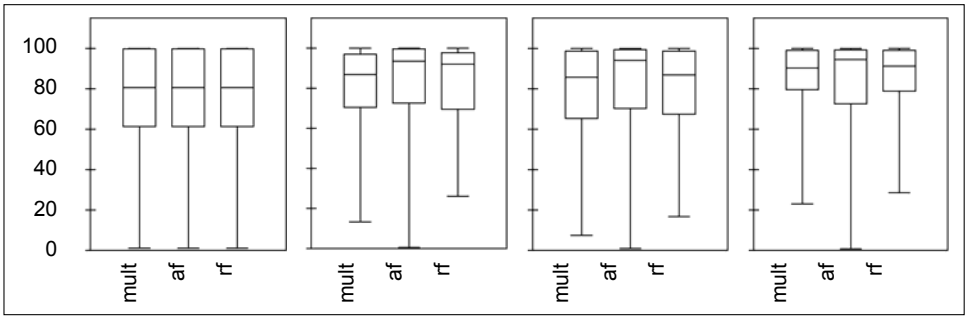


Рис. 3. Метод **fn-nofb** со шкалами, слева направо: **unit**/**unit**, **orig**/**orig**, **buck-lin**/**buck-lin** и **buck-exp**/**buck-exp**

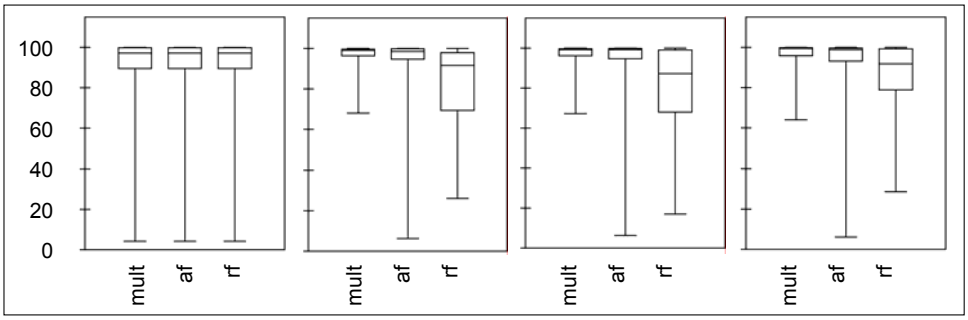


Рис. 4. Метод **fn-fb** со шкалами, слева направо: **unit**/**unit, orig**/**orig, buck-lin**/**buck-lin** и **buck-exp**/**buck-exp**

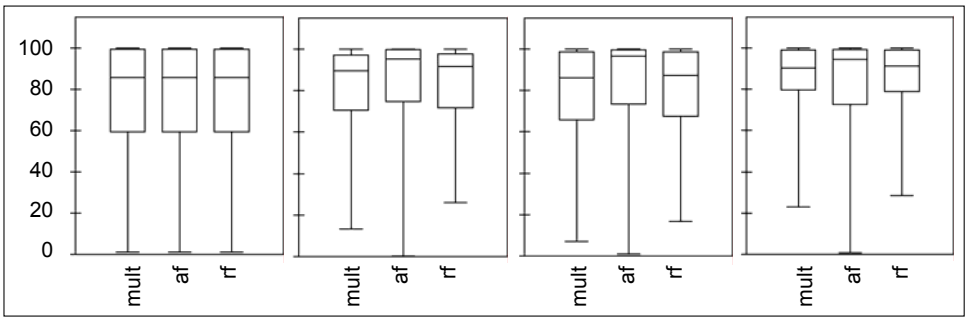


Рис. 5. Метод **diff-fn-nofb** со шкалами, слева направо: **unit**/**unit, orig**/**orig, buck-lin**/**buck-lin** и **buck-exp**/**buck-exp**

Исследование 2. В данном исследовании, было изучено влияние шкалы стоимости тестов и серьезности ошибок на относительную эффективность методов приоритезации. Для каждого из трех методов комбинации

информации и четырех комбинаций шкал (**unit/unit, orig/orig, buck-lin/buck-lin, buck-exp/buck-exp**) были сравнены друг с другом пять методов приоритезации. Подобно предыдущему исследованию, был проведен статистический анализ Anova. В каждом из двенадцати случаев, Anova анализ показал статистическую значимость различий между методами приоритезации. Наименьшее значение $p = 0{,}0263$ было достигнуто **rf** комбинацией и **buck-exp/buck-exp** шкалой.

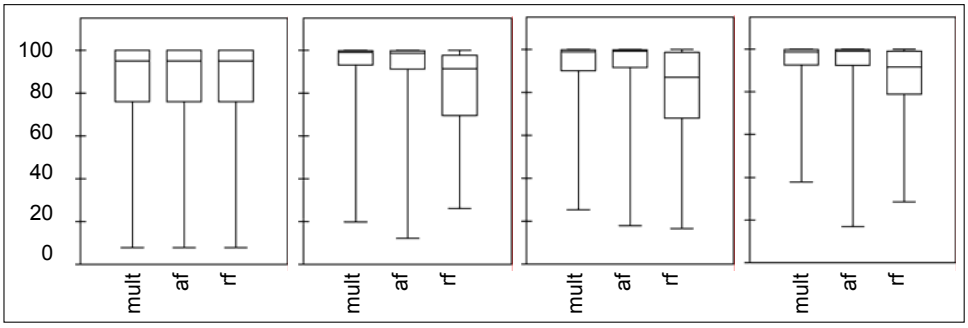


Рис. 6. Метод **diff-fn-fb** со шкалами, слева направо: **unit/unit, orig/orig, buck-lin/buck-lin** и **buck-exp/buck-exp**

Как мы видим из графиков (рис. 7–9) (представленных диаграммами размаха), с точки зрения среднего значения метрики $APFD_C$, при комбинации **af** шкала имеет незначительное влияние на относительную эффективность методов приоритезации. За исключением шкалы **orig/orig**, методы приоритезации в порядке убывания эффективности следуюие: **fn-fb, fn-diff-fb, random, fn-diff-nofb, fn-nofb**. Также, с точки зрения среднего значения метрики $APFD_C$, при комбинации **mult,** шкала вовсе не имеет влияния на относительную эффективность методов приоритезации. С точки зрения среднего значения метрики $APFD_C$, при комбинации **rf**, шкала имеет более значительное влияние на относительную эффективность методов приоритезации. При данной комбинации, методы приоритезации имеют незначительное различие в их эффективности; гораздо больший вклад в их эффективность имеет шкала. В шкалах **orig/orig, buck-lin/buck-lin** и **buck-exp/buck-exp**, **random** метод был лучше, чем остальные методы приоритезации, что означает бесполезность приоритезации в данных случаях. С другой стороны, в шкале **unit/unit** стоимость фактически не учитывается и относительная эффективность методов приоритезация такая же как и в других комбинациях.

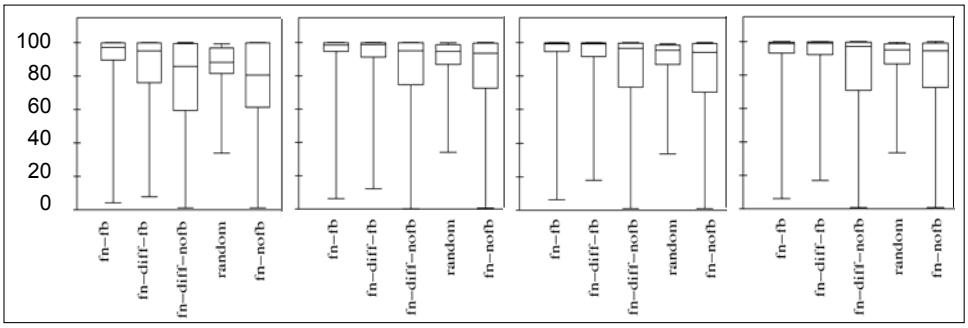


Рис. 7. **Af** комбинация со шкалами, слева направо: **unit/unit, orig/orig, buck-lin/buck-lin** и **buck-exp/buck-exp**

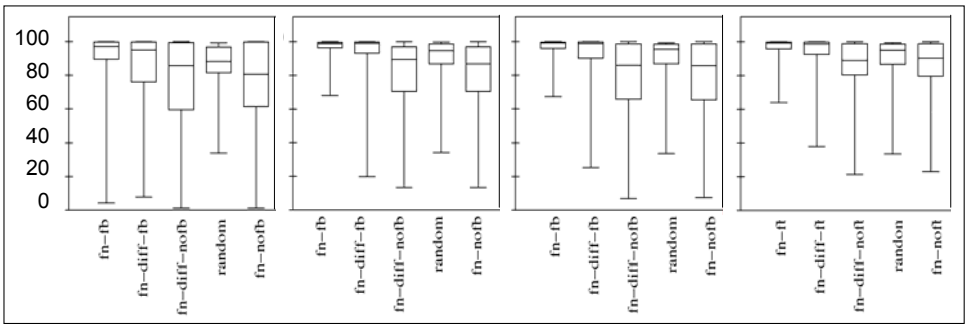


Рис. 8. **Mult** комбинация со шкалами, слева направо: **unit/unit, orig/orig, buck-lin/buck-lin** и **buck-exp/buck-exp**

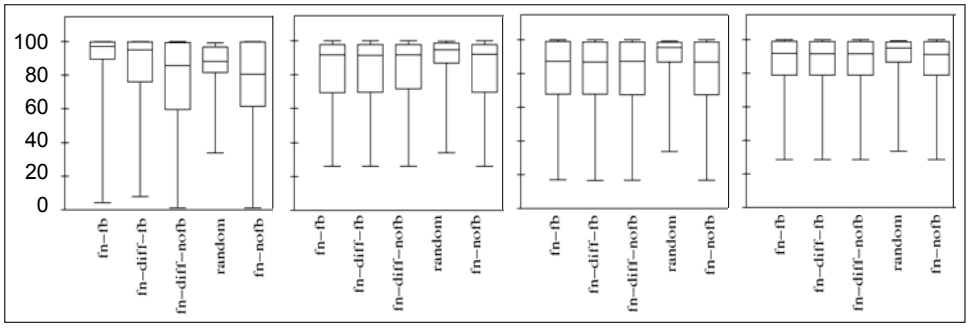


Рис. 9. **Rf** комбинация со шкалами, слева направо: **unit/unit, orig/orig, buck-lin/buck-lin** и **buck-exp/buck-exp**

Исследование 3. В данном исследовании было рассмотрено для четырех методов приоритезации и трех методов комбинации все 16 шкал. Для метода приоритезации **fn-fb** и комбинации **af**, различия в средних значениях $APFD_C$ не являются статистически значимыми ($p = 0{,}4313$). Во всех других одиннадцати случаях, различия статистически значимы (максимальное значение $p = 0{,}0015$).

Как видно из графиков (рис. 10–13) представленных диаграммами размаха, для метода **fn-nofb**, размах средних значений $APFD_C$ для различных шкал максимален для **rf** — от 74,92 (**unit/orig**) до 86,55 (**orig/unit**); затем идет **mult** — от 75,81 (**unit/unit**) до 85,84 (**buck-exp/buck-exp**); и наименьший размах для **af** — от 74,54 (**buck-exp/unit**) до 81,82 (**unit/buck-exp**). Для метода **fn-fb**, размах средних значений небольшой при комбинациях **af** и **mult** (меньше 5), но значителен для **rf** — от 75,04 (**unit/orig**) до 92,71 (**unit/unit**). Для метода **diff-fn-nofb**, размах максимален при **rf** — от 74,94 (**unit/orig**) до 86,85 (**orig/unit**), затем идет **mult** — от 76,07 (**buck-lin/unit**) до 85,62 (**buck-exp/buck-exp**); и наименьший размах при **af** — от 76,20 (**orig/unit**) и 84,15 (**buck-exp/buck-exp**). Для метода **diff-fn-fb**, размах максимален при **rf** — от 74,93 (**unit/orig**) до **orig/unit** (86,67); затем идет **af** — от 85,01 (**orig/unit**) до 91,29 (**buck-exp/buck-lin**); наименьший размах имеет место при **mult** — от 86,19 (**unit/unit**) до 92,62 (**buck-exp/buck-lin**).

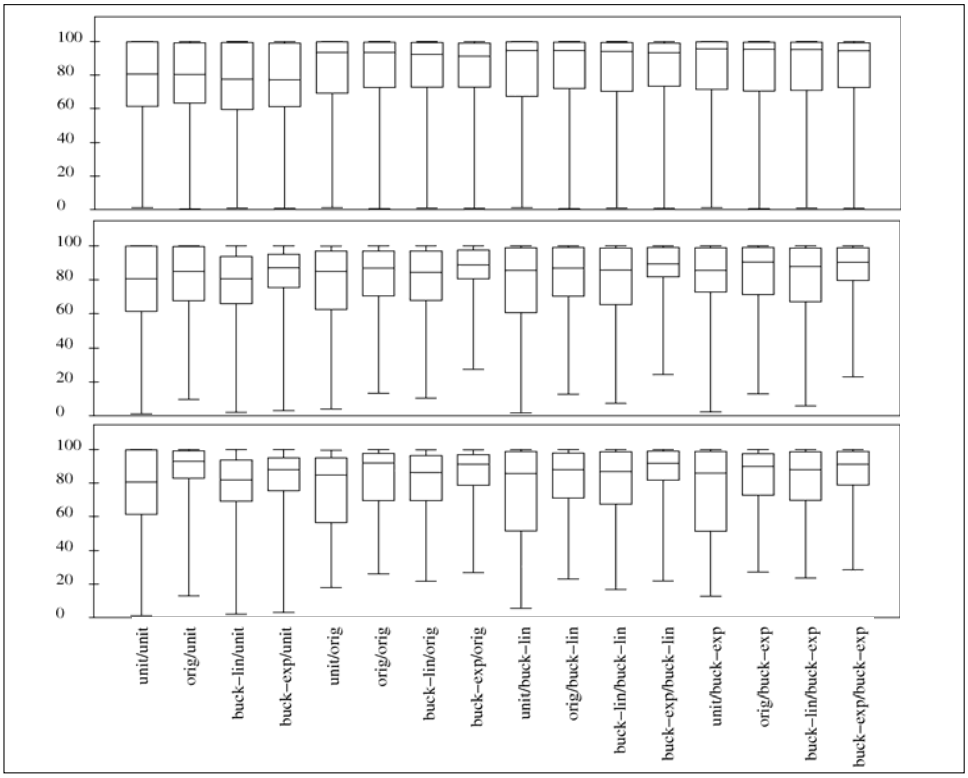


Рис. 10. Метод **fn-nofb** с комбинациями, сверху вниз: **af, mult** и **rf**

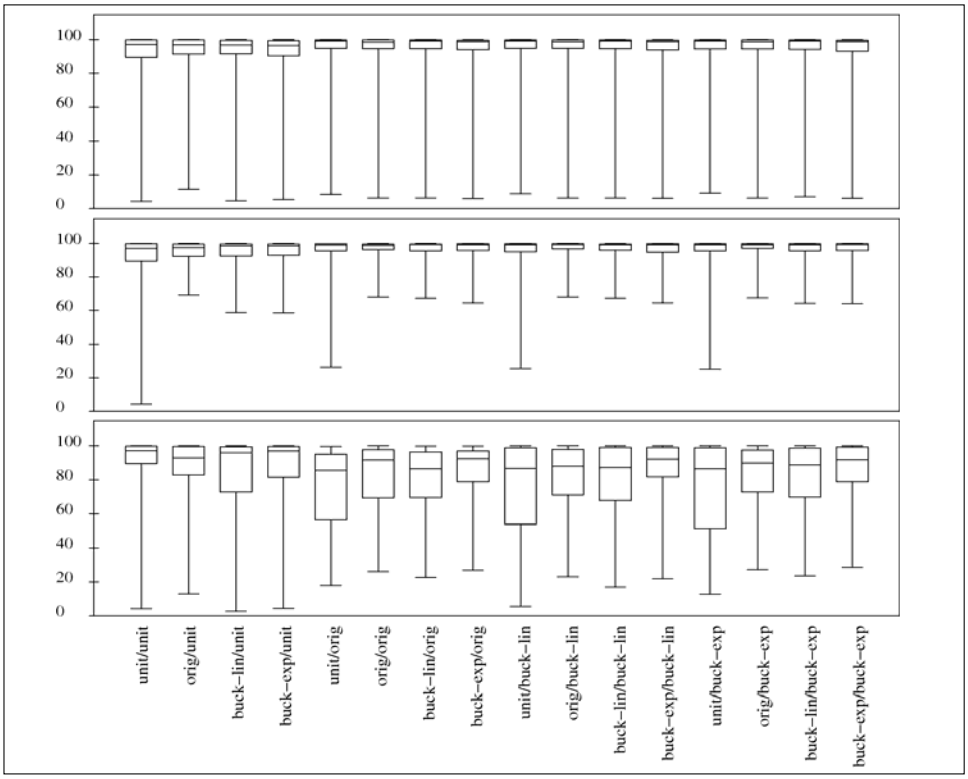


Рис. 11. Метод **fn-fb** с комбинациями, сверху вниз: **af, mult** и **rf**

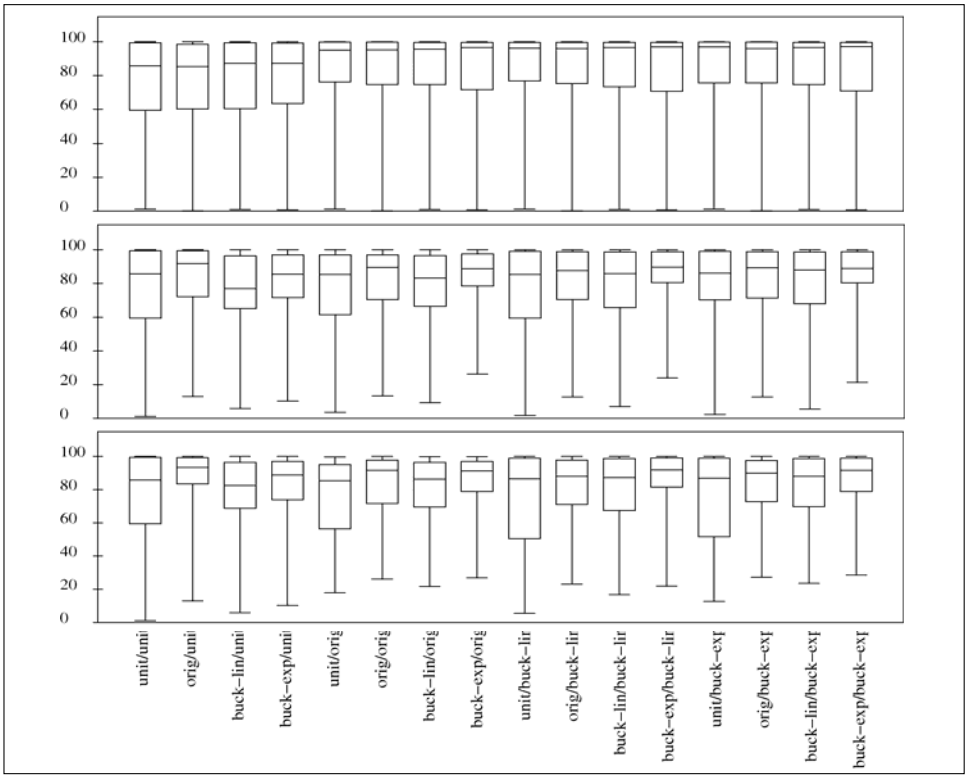


Рис. 12. Метод **diff-fn-nofb** с комбинациями, сверху вниз: **af, mult** и **rf**

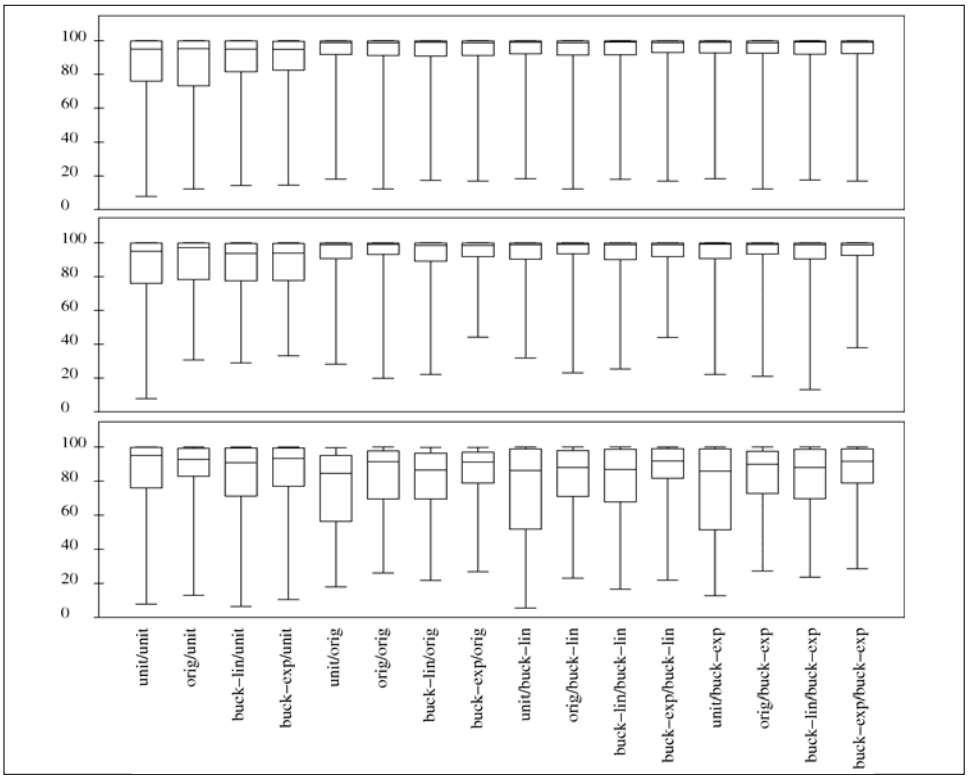


Рис. 13. Метод **diff-fn-fb** с комбинациями, сверху вниз: **af, mult** и **rf**

ВЫВОДЫ

В данной работе была рассмотрена приоритезация, учитывающая стоимость тестов и серьезность ошибок. В отличие от предыдущих исследований, серьезность ошибок была основана на их влиянии на надежность программы. Несколько программ были использованы как объекты исследования. В одной из них были использованы реальные данные о стоимости тестов и реальный операционный профиль. Также, как стоимость тестов, так и серьезность ошибок были исследованы в четырех различных шкалах и изучено их влияние на эффективность приоритезации. Были рассмотрены три подхода к внедрению информации о стоимости и критичности тестов в процесс приоритезации и исследована их эффективность.

Выбор шкалы имеет значительное влияние на эффективность приоритезации. Важно то, что выбор лучшего метода приоритезации зависит от шкалы стоимости тестов и серьезности ошибок. Разные шкалы неодинаково влияют на разные методы приоритезации. Поэтому выбор метода приоритезации должен учитывать шкалу. Также, методы комбинации **af** и **mult**, а особенно **af**, дают наилучшие результаты.

Автор благодарит Г. Ротермела, С. Элбаума и Д. Рутруфа за участие в проведении описанных исследований.

ЛИТЕРАТУРА

1. *Ghezzi C., Jazayeri M., Mandrioli D.* Fundamentals of Software Engineering. — Upper Saddle River: Prentice Hall, 1991. — 573 p.
2. *Avritzer A., Weyuker E.J.* The automatic generation of load test suites and the assessment of the resulting software // IEEE Transactions on Software Engineer. — 1995. — **21**, № 9. — P. 705–716.
3. *Wong W., Horgan J., London S., Agrawal H.* A study of effective regression testing in practice // In Proceedings of the Eighth International Symposium on Software Reliability Engineering. — Albuquerque, NM, USA. — 1997. — P. 230–238.
4. *Rothermel G., Untch R., Chu C., Harrold M.J.* Test case prioritization: an empirical study // In Proceedings of the International Conference on Software Maintenance. — Oxford, England, UK. — 1999. — P. 179–188.
5. *Elbaum S., Malishevsky A., Rothermel G.* Prioritizing test cases for regression testing // In Proceedings of the International Symposium on Software Testing and Analysis. — Portland, Oregon. — 2000. — P. 102–112.
6. *Elbaum S., Malishevsky A., Rothermel G.* Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization // Technical Report 00-60-09. — Computer Science Department. — Oregon State University. — August 2000. — 14 p.
7. *Rothermel G., Untch R.H., Chu C., Harrold M.J.* Test case prioritization // IEEE Transactions on Software Engineering. — 2001. — **27**, № 10. — P. 929–948.
8. *Jones J.A., Harrold M.J.* Test-suite reduction and prioritization for modified condition/decision coverage // In Proceedings of the International Conference on Software Maintenance. — Florence, Italy. — 2001. — P. 92–101.
9. *Elbaum S., Malishevsky A., Rothermel G.* Incorporating varying test costs and fault severities into test case prioritization // In Proceedings of the 23rd International Conference on Software Engineering. — Toronto, Ontario, Canada. — May 2001. — P. 329–338.

10. *Elbaum S., Malishevsky A., Rothermel G.* Test Case Prioritization: A family of empirical studies // IEEE Transactions On Software Engineering. — 2002. — **28**, № 2. — P. 159–182.
11. *Srivastava A., Thiagarajan J.* Effectively prioritizing tests in development environment // In Proceedings of the International Symposium on Software Testing and Analysis. — Via di Ripetta, Rome – Italy. — 2002. — P. 97–106.
12. *Kim J.-M., Porter A.* A history-based test prioritization technique for regression testing in resource constrained environments // In Proceedings of the International Conference on Software Engineering. — Orlando, Florida, USA. — 2002. — P. 119–129.
13. *Malishevsky A.G.* Test case prioritization // Ph.D. Dissertation. — Oregon State University, Corvallis, Oregon, USA. — 2003. — 291 p.
14. *Do H., Rothermel G. and Kinneer A.* Empirical Studies of Test Case Prioritization in a JUnit Testing Environment // Proceedings of the International Symposium on Software Reliability Engineering. — Saint-Malo, Bretagne, France. — November 2004. — P. 113–124.
15. *Srikanth H.* Value-driven system level test case prioritization // Ph.D. Dissertation. — North Carolina State University, Releigh, NC. — 2005. — 92 p.
16. *Korel B., Tahat L.H. and Harman M.* Test Prioritization Using System Models // In Proceedings of the 21st IEEE International Conference on Software Maintenance. — Budapest, Hungary. — September 2005. — P. 559–568.
17. *Do H. and Rothermel G.* A Controlled Experiment Assessing Test Case Prioritization Techniques via Mutation Faults // Proceedings of the IEEE International Conference on Software Maintenance. — Budapest, Hungary. — September 2005. — P. 411–420.
18. *Do H., Rothermel G. and Kinneer A.* Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis // Empirical Software Engineering: An International Journal. — March 2006. — **11**, № 1. — P. 33–70.
19. *Do H. and Rothermel G.* On the use of mutation faults in empirical assessments of test case prioritization techniques // IEEE Transactions on Software Engineering. — September 2006. — **32**, № 9. — P. 733–752.
20. *Малышевский А.Г.* Приоритезация тестов в регрессивном тестировании // Системні дослідження та інформаційні технології. — 2006. — № 4. — С. 16–32.
21. *Malishevsky A.G., Ruthruff J., Rothermel G. and Elbaum S.* Cost-cognizant test case prioritization // Technical Report TR-UNL-CSE-2006-0004. — Department of Computer Science and Engineering. — University of Nebraska — Lincoln. — March 2006. — 41 p.
22. *Малышевский А.Г.* Использование информации о стоимости тестов и серьезности ошибок в процессе приоритезации тестов // Системні дослідження та інформаційні технології. — 2008. — № 1. — С. 63–78.
23. *Musa J.* Software Reliability Engineering. — NY: McGraw-Hill, 1999. — 391 p.

Поступила 27.07.2008