Web-application testability evaluation

Alexei Barantsev
Institute for System
Programming of RAS
email: barancev@ispras.ru

Abstract (English)

This article introduces new aspects to the notion of web-application testability. Review of existing investigations related to software testability is provided. Restrictions of existing approaches and ways to overcome these restrictions are discussed. The article provides preview of a novel web-application testability evaluation tool.

***Keywords**: Software testing; software testability; software quality; software maintainability; web-applications.*

Оценка тестопригодности веб-приложений

Баранцев Алексей
Институт системного
программирования РАН
email: barancev@ispras.ru

Abstract (Russian)

Данная работа вводит новые аспекты в рассмотрение понятия тестопригодности веб-приложений. Приводится обзор работ, в которых исследуется понятие тестопригодности программ; обсуждаются недостатки существующих подходов, после чего предлагаются пути преодоления этих недостатков; приводится описание прообраза инструмента, реализующего предложенные идеи применительно к веб-приложениям.

***Keywords**: Тестирование программ; тестопригодность программ; качество программ; сопровождамость программ; веб-приложения.*

1. Введение. Обзор работ

Существующие работы можно разбить на две группы, в которых понятие тестопригодности рассматривается в одном из двух контекстов – реализационном либо спецификационном. Спецификационный контекст допускает более формальный подход к определению тестопригодности, поскольку это область, в которой можно работать с математическими моделями программ. Реализационный контекст ближе к практике, здесь исследователям приходится больше опираться на эмпирические знания, поэтому в этом контексте превалирую т неформальные определения тестопригодности.

Обзор работ частично опирается на статьи Баумгартена [1] и Брунтинка и Дёрсена [3]. Баумгартен рассматривал исключительно вопросы тестопригодности спецификаций, оставляя в стороне реализационные аспекты. Брутник и Дёрсен, напротив, больше внимания уделяли практической стороне дела.

Первая попытка формально определить понятие тестопригодности принадлежит, видимо, Карнапу [4]. Разумеется, никакого отношения к тестированию программ работы Карнапа не имели, он пытался определить, какие логические утверждения можно считать проверяемыми (testable), а какие нельзя. Тем не менее, современные исследования тестопригодности продолжают заложенную Карнапом традицию.

Кальман в работах по теории управления [7] сформулировал определение тестопригодности для детерминированных конечных автоматов. Он определил тестопригодность как комбинацию двух других свойств:

- *управляемость (controllability) – возможность достигнуть любого состояния, подавая на вход различные стимулы,*
- *наблюдаемость (observability) – возможность определения состояния системы путём подачи на вход определённых стимулов и наблюдения полученных в ответ на эти стимулы реакций.*

Чуть позже Штарк [9] расширил понятие наблюдаемости так, что оно стало применимо также для недетерминированных конечных автоматов.

Неформальные обсуждения понятия тестопригодности программ можно найти в [2], [5], [10]. В этих работах тестопригодность трактуется как наличие у программы тех или иных характеристик, облегчающих её тестирование. При этом «облегчение» может выражаться либо в сокращении стоимости, времени или усилий, либо в возможности применения тех или иных методов тестирования или инструментальных средств.

Принимая во внимание тот факт, что тестирование является инженерной, а не чисто теоретической дисциплиной, практические ограничения, такие как стоимость или время имеют не меньшее значение, чем теоретическая возможность создания набора тестов.

Поэтому усилия исследователей нацелены на то, чтобы для распространённых частных случаев определить некоторые совокупности характеристик программ и свести их к некоторому набору метрик, которые могут быть вычислены на основе реализации, и для которых имеется корреляция с полученными эмпирическим путём оценками затраченных на тестирование усилий.

Фридман [5], так же как и Кальман [7], определил понятие тестопригодности для программ как сочетание управляемости и наблюдаемости. Однако для управляемости и наблюдаемости он дал более широкие определения в терминах входных и выходных значений, поскольку программу не всегда можно трактовать как конечный автомат. Согласно Фридману,

- *управляемость определяет, какую часть области выходных значений можно покрыть, подавая на вход все возможные варианты входных значений,*
- *наблюдаемость характеризуется тем, получаем ли мы на выходе разные или одинаковые значения на выходе, подавая на вход те или иные данные.*

Если представить себе программу как функцию из области входных данных в область выходных данных, то можно сказать, что инъективная функция обладает хорошей наблюдаемостью, а сюръективная функция – хорошей управляемостью.

Воас и Миллер [10] также рассматривали отношение мощности множества выходных данных к мощности множества входных данных. Они изучали связь этого показателя с такой характеристикой, как степень чувствительности программы к наличию дефектов, то есть возможность обнаружения дефекта в программе, если известно, что он там имеется.

Другие исследователи пытались опираться не на «внешние», а не «внутренние» метрики, которые могут быть вычислены на базе исходного кода программы. Джунгмейр [6] изучал влияние связей между частями программы на сложность интеграционного тестирования. МакГрегор [8] аналогичное исследование проводил в отношении связей, характерных для программ, написанных на объектно-ориентированных языках – наследования, делегирования, исключений.

2. Тестопригодность веб-приложений

Все изученные нами работы неявно предполагают два фактора, которые, вообще говоря, нельзя принимать за истину:

- тестопригодность зависит только от тестируемой программы,
- интерфейс взаимодействия тестовой системы с тестируемой программой не влияет на тестопригодность.

В данной главе мы рассматриваем эти два фактора, показываем, как можно их учесть, а в следующей главе описывается прообраз инструмента, предназначенного для оценки тестопригодности веб-приложений с учётом этих двух факторов.

2.1. Зависимость от инструментов, целей и методов тестирования

Когда исследователи пытаются определить тестопригодность в терминах метрик, присущих самой программе, они полагают, что тестопригодность можно рассматривать как некую функцию программы:

$$Testability = F(Program)$$

Как было показано выше, исследователи пытаются свести эту функцию к вычислению определённых метрик, базирующихся либо на исходном коде программы, либо на области входных и выходных значений. Вряд ли у кого-либо вызовет сомнения тот факт, что сложность программы кореллирует с усилиями, которые необходимы для её разработки, тестирования и поддержания. Однако определение сложности программы не так-то просто свести к измерению тех или иных характеристик исходного кода.

Поэтому такой способ относительно хорошо работает для простых случаев – модульное функциональное тестирование, и практически неприменим в других случаях.

Действительно, если посмотреть на типичное веб-приложение, можно видеть, что часть функциональности реализуется на стороне браузера на языке JavaScript или его аналоге, часть на сервере приложений, и часть – в базе данных на языках PL/SQL, TransactSQL или подобных им. Нам не удалось найти ни одной работы, в которой приводился бы способ оценки тестопригодности на основе анализа исходного кода для приложений, представляющих собой подобную совокупность разнородных частей.

Кроме того, все встреченные нами работы были посвящены оценке пригодности программы для функционального тестирования. Можно предположить, что анализ программного кода не позволяет оценить пригодность программы для других видов тестирования – производительности, удобства использования, защищенности и т.д.

Нам кажется, что эти исследователи в своих работах неявно предполагали и поэтому не упоминали про одну крайне важную вещь: тестирование всегда нацелено на достижение определённых целей. Эти цели могут быть различными, что и порождает разные виды или подвиды тестирования. Так, при тестировании производительности и функциональности ставятся разные цели. Аналогично, при модульном и системном тестировании цели также различаются. То есть тестопригодность является функцией не только тестируемой программы, но также цели тестирования:

Testability = F(Program, Target)

Но и это ещё не всё. Чтобы понять, чего не хватает, проведём ещё одну аналогию, поговорим о достижимости некоторого места. Достижим ли Китай? Достижим ли необитаемый остров посреди Тихого океана? Достижимо ли дно Марианской впадины? Достижима ли обратная сторона Луны? Для ответа на этот вопрос нужно знать, какими техническими средствами мы располагаем. Когда-то на любой из этих вопросов ответ был однозначный – нет, но по мере развития техники эти места стали достижимыми.

Точно так же тестопригодность зависит не только от самой программы, но также и от имеющихся в нашем распоряжении инструментальных средств:

Testability = F(Program, Target, Tool)

Чуть ниже мы подробнее обсудим эту зависимость тестопригодности от инструментальных средств на примере веб-приложений. А пока отметим только, что в каждом конкретном случае нужно принимать во внимание не абстрактный «уровень технологического развития человечества», но конкретные инструменты, которые либо уже имеются, либо могут быть приобретены, не выходя за рамки доступного бюджета.

Следует также иметь в виду, что инструменты могут поддерживать несколько различных методов тестирования. Так, например, для веб-приложений возможно тестирование как на уровне браузера (то есть эмуляции действий пользователя с клавиатурой и мышью), так и на уровне HTTP-протокола. Поскольку для программы использование одного способа может оказаться проще, чем использование другого, а также инструменты могут лучше поддерживать тот или иной способ и хуже другие способы, мы явно подчеркнём эту зависимость от выбранного способа тестирования:

Testability = F(Program, Target, Tool, Method)

Таким образом, можно сказать, что в большинстве имеющихся работ изучается частный случай этой функции, когда три последних параметра фиксированы, а именно: целью является модульное функциональное тестирование с оценкой полноты покрытия программного кода, используются традиционные инструменты модульного тестирования, способ тестирования – методом «чёрного» либо «белого» ящика на уровне программных интерфейсов (API).

Разумеется, мы не претендуем в данной работе полностью описать указанную функцию для всех возможных случаев. Мы всего лишь ставим целью показать, что имеется большой простор для развития исследований не только вглубь, но и вширь.

Вернёмся ещё раз к аналогии с понятием достижимости какого-либо места. Для повышения достижимости у нас могут быть два пути – либо совершенствование транспортных средств, либо улучшение инфраструктуры, которое позволит использовать уже существующие транспортные средства. Так, чтобы обеспечить возможность доставки грузов в труднодоступное место, можно либо купить (а может быть даже изобрести) вертолёт, либо построить железную или автомобильную дорогу.

Точно так же можно говорить о повышении тестопригодности программ, причём тут тоже можно действовать с двух сторон – либо делать более хитроумные инструменты тестирования, либо разрабатывать приложения с учётом тестопригодности для уже имеющихся инструментов.

Опираясь на вышеприведённую формулу тестопригодности:

Testability = F(Program, Target, Tool, Method)

можно видеть, что при этом мы фиксируем два параметра – цель и способ тестирования, и варьируем один из оставшихся параметров (или оба сразу).

2.2. Тестопригодность интерфейса

Второй серьёзный недостаток имеющихся работ проистекает из того, что в них рассматривается тестирование на уровне программных интерфейсов. При таком предположении можно считать, что мы можем подавать на вход любые допустимые спецификацией интерфейса стимулы, а также можем без труда наблюдать реакции тестируемой программы.

Однако на практике это предположение часто не выполняется. Например, при тестировании программ с графическим пользовательским интерфейсом, как подача стимулов, так и наблюдение реакций представляют собой отдельную технически весьма сложную задачу. При тестировании компонентов, которые должны работать в определённом окружении, например, сервлетов или EJB, зачастую используют искусственное окружение (псевдо-контейнер).

Поэтому, в дополнение к общему понятию тестопригодности программы, можно ввести более узкое понятие тестопригодности того или иного интерфейса, имея в виду управляемость – возможность и простота посылки стимулов через этот интерфейс, и наблюдаемость – возможность и простота наблюдения реакций.

Для веб-приложений широко используется два способа функционального тестирования – на уровне HTTP-запросов и на уровне браузера.

Первый способ хорош тем, что тестовая система получается независима от браузера и чаще всего даже от операционной системы. Но в современных веб-приложениях зачастую часть логики реализуется в виде скриптов на языке JavaScript, которые исполняются в браузере. Поэтому такой способ применим только к программам, в которых вся логика реализована на серверной стороне.

Второй способ требует достаточно тесной интеграции инструментов тестирования с различными браузерами и способности инструментов эмулировать различные действия пользователя.

С точки зрения простоты отправки стимулов и наблюдения реакций интерфейс HTTP-запросов обладает лучшими показателями, формирование и анализ HTTP-запросов и ответов на них является технически более простой задачей, чем эмуляция действий пользователя в браузере – нажатий кнопки мыши и её движений, нажатия клавиш на клавиатуре.

Однако с точки зрения возможности отправки всех возможных стимулов и наблюдения всех возможных реакций при наличии скриптов, исполняемых в браузере, первый способ явно проигрывает второму, несмотря на увеличивающуюся при этом техническую сложность.

Для повышения тестопригодности интерфейса, как уже отмечалось выше, можно предпринимать усилия с двух сторон – либо проектировать приложение так, чтобы инструментам было проще получить доступ к элементам интерфейса, либо создавать инструменты, способные работать с произвольными интерфейсами. Покажем это на конкретном примере.

В литературе можно найти советы по проектированию веб-интерфейсов, нацеленные на повышение тестопригодности, среди которых чаще всего встречается рекомендация использовать статическое именование структурных элементов, то есть указание значений атрибутов id или name, которые заранее определены, а не генерируются динамически и не меняются впоследствии при модификации приложения. Однако в последнее время можно заметить, что инструменты стали более мощными, использование для идентификации элементов интерфейса языка запросов XPath позволило практически полностью отказаться от статического именования этих элементов.

3. Прообраз реализации

В настоящее время в Институте системного программирования РАН разрабатывается инструмент для оценки тестопригодности веб-приложений, который учитывает два рассмотренных выше фактора.

Инструмент предназначен для того, чтобы на ранних стадиях проекта по созданию веб-приложения, когда уже готов статический дизайн страниц, но приложение ещё находится в разработке, помочь выбрать наиболее подходящий для тестирования инструмент и способ тестирования, либо выдать рекомендации по доработке интерфейса веб-приложения, так чтобы можно было воспользоваться имеющимися инструментами тестирования.

Для поддержки работы этого инструмента создана информационная система, в которой описываются поддерживаемые различными инструментами цели и способы тестирования, а также анализируются и фиксируются

ограничения инструментов тестирования, такие как:

- возможность работы на уровне HTTP-протокола,
- интеграция с различными браузерами,
- возможность поддержки сессий,
- возможность интерпретации JavaScript,
- возможность работы с диалоговыми окнами,
- поддержка различных способов аутентификации, и т.д.

Инструмент анализирует структуру страниц веб-приложения и определяет, насколько хорошо тот или иной инструмент сможет работать с этим интерфейсом, после чего сообщает, какие проблемы могут возникнуть при использовании тех или иных инструментов.

Это даёт возможность более осознанно подбирать способы и инструменты тестирования, что положительно сказывается на качестве программы и сокращает расходы на разработку.

4. Список литературы

[1] B. Baumgarten, H. Wiland. Qualitative Notions of Testability. IWTCS 1998: 345-360

[2] B.Beizer. Software Testing Techniques. 2nd ed. V. Nostrand Reinhold, 1990

[3] M. Bruntink, A. van Deursten. Predicting Class Testability using Object-Oriented Metrics. 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04)

[4] R. Carnap. Testability and Meaning. Philosophy of Science, 3/4, pp 420-471, 4/1, pp1-40, 1936/37

[5] R.S. Fridman. Testability of Software Components. IEEE Transactions on Software Engineering, 17/6, pp553-564, 1991

[6] S. Jungmayr. Identifying test-critical dependencies. In Proceedings of the International Conference on Software Maintenance, pages 404–413. IEEE Computer Society, October 2002

[7] R.E. Kalman, P.L. Falb, M.A. Arbib. Topics in Mathematical System Theory. McGraw-Hill, 1969

[8] J. McGregor, S. Srinivas. A measure of testing effort. In Proceedings of the Conference on Object-Oriented Technologies, pages 129–142. USENIX Association, June 1996

[9] P.H. Starke. Abstrakte Automaten. Deutscher Verlag der Wissenschaften, 1969

[10] J.M. Voas, K.W. Miller. Software Testability: The New Verification. IEEE Software, May 1995