Энтропийно-информационный анализ эффективности модульного тестирования программных средств

Г. И. Кожомбердиева[1], Д. П. Бураков[2]
Петербургский государственный университет путей сообщения Императора Александра I
[1]kgi-liizht@yandex.ru, [2]burakovdmitry8@gmail.com

***Аннотация*. В докладе тестирование программного кода рассматривается как информационный процесс, в котором каждый пройденный (с возможным последующим исправлением ошибки и повторением запуска) тест уменьшает неопределенность и увеличивает уверенность разработчика относительно работоспособности кода. Показана возможность применения энтропийно-информационного анализа модульного тестирования для получения формальной оценки его эффективности. Энтропия используется в качестве меры неопределенности относительно работоспособности программного кода в составе исполняемой программы, как при отсутствии, так и при выполнении модульного тестирования кода. Полученное авторами выражение для вычисления эффективности модульного тестирования может использоваться как одна из метрик важнейшей характеристики качества программного обеспечения − надежности. Данный метрический показатель не только хорошо согласуется с такой широко применяемой метрикой, как тестовое покрытие, но и уточняет ее.**

Ключевые слова: энтропийно-информационный анализ; неопределенность; энтропия; модульное тестирование; программный код; тестовое покрытие; дефект кода

I. Введение

Разработка программного продукта представляет собой сложный процесс, как правило, разбиваемый на отдельные последовательные этапы: формирование концепции системы, системных требований и требований к программному обеспечению; разработка архитектуры системы, ее разбиение на модули; разработка отдельных модулей, реализация функций системы на языке программирования. По мере реализации проекта в целях обеспечения качества создаваемого программного изделия предприятие-разработчик выполняет работы по *верификации* − проверке соответствия промежуточных продуктов установленным требованиям. Конечный продукт программного проекта подвергается процедурам *валидации*, обеспечивающим проверку полного соответствия изделия требованиям и гарантирующим его безопасное и надежное применение пользователем. Следует подчеркнуть, что гарантиями качества перед потребителем программного продукта выступают как результаты контроля соответствия характеристик качества установленным требованиям, так и достигнутый предприятием-разработчиком уровень зрелости процессов разработки программного обеспечения (к которым относятся, в частности, сами процессы верификации и валидации) [1 – 3].

Верификация как процесс обеспечения заданного уровня качества создаваемого программного продукта включает в себя различные формы инспекционного контроля, формирование и анализ отчетной документации, но основным содержанием верификации является *тестирование* − процесс выполнения программного кода с целью выявления *дефектов*. При этом под дефектом понимается участок кода, выполнение которого при определенных условиях приводит к поведению, не соответствующему требованиям, нарушению работоспособности. Выяснение природы обнаруженной ошибки и ее устранение осуществляется в процессе *отладки* программы.

Не вызывает сомнения, что тестирование можно рассматривать как информационный процесс, в котором каждый пройденный (с возможным последующим исправлением ошибки и повторением запуска) тест уменьшает неопределенность и увеличивает уверенность разработчика относительно работоспособности программного кода. В этой связи авторам доклада показалось интересным выполнить энтропийно-информационный анализ [4, 5] тестирования с целью получения формальной оценки его эффективности. Полученное авторами выражение для вычисления эффективности тестирования может использоваться как одна из метрик важнейшей характеристики качества программного обеспечения − *надежности*.

В программной инженерии отдельно рассматриваются уровни системного, интеграционного и модульного тестирования. В настоящем докладе авторы ограничиваются анализом уровня модульного тестирования.

II. О модульном тестировании

Модульное тестирование (англ. *unit testing*) − это испытания программы на уровне отдельных модулей, проверка каждого программного модуля (например, библиотеки функций в программе на процедурном языке, класса в программе на объектно-ориентированном языке) на наличие дефектов. Целью модульного тестирования

является выявление простых ошибок в реализации алгоритмов функций или методов класса, в использовании локальных переменных, а также определение степени готовности программного продукта к переходу на следующий уровень разработки и тестирования. При выполнении модульного тестирования используется *метод «белого ящика»*, т.е. тестирование данного уровня основывается на знании внутренней реализации программы.

При unit-тестировании на конечном наборе тестов проверяется соответствие между фактическим и ожидаемым поведением программного модуля. *Тестовое покрытие* (англ. *statement coverage*) – это одна из метрик оценки качества тестирования, которая показывает степень покрытия тестами исполняемого кода. Чем выше требуемый уровень тестового покрытия, тем больше должно быть тестов в наборе [1, 2, 6].

Процесс создания unit-тестов автоматизирован, причем при написании тестов и тестируемого кода используется один и тот же язык программирования. Тесты разрабатывает сам программист, занимающийся написанием или отладкой соответствующих модулей. Unit-тесты, помимо применения в процессе верификации, полезно использовать в следующих случаях:

- В технологии TDD (англ. *Test-Driven Development* – разработка программного кода, управляемая тестами), которая предполагает создание тестов, определяющих требования к разрабатываемой функции (методу) до написания кода функции (метода), позволяющего пройти данные тесты.
- При проведении *рефакторинга* объектно-ориентированного исходного кода, т.е. выполнении изменения кода с целью облегчения понимания его работы при сохранении функциональности. Запуск unit-тестов после внесения изменений позволяет отследить появление возможных ошибок.

Для автоматизации создания, запуска и отслеживания результатов работы unit-тестов, управления выполнением серий тестов используются специальные системы программного обеспечения − *фреймворки тестирования* (англ. *framework* − каркас, структура). Примером такого продукта является *JUnit* − фреймворк тестирования на платформе Java. Интегрированные среды разработки Java-приложений, как правило, имеют стандартные подключаемые модули для поддержки *JUnit* [6 − 10].

III. Энтропийно-информационный подход к оценке эффективности модульного тестирования

Будем рассматривать тестирование в качестве информационного процесса, в котором каждый пройденный тест уменьшает неопределенность и увеличивает уверенность разработчика относительно работоспособности программного кода. Такая интерпретация позволяет предложить вероятностную модель выполнения кода как в реальной среде в составе исполняемой программы, без предварительных тестовых испытаний, так и в среде, формируемой и предоставляемой системой тестирования. Представленная ниже модель используется для получения соответствующих обоим случаям оценок энтропии как меры неопределенности.

1. Будем предполагать, что модуль, работающий в составе программы в реальной среде, состоит из N строк кода, в каждой из которых с некоторой вероятностью p_i, $i = 1, \ldots, N$ содержится дефект, который может привести к ошибке. Строго говоря, вероятности p_i могут быть различны для разных строк кода в зависимости от сложности операции, выполняемой в данной строке и от квалификации программиста, разрабатывающего модуль. Кроме того, число реально выполняемых строк кода может отличаться от N при наличии в теле модуля операторов изменения последовательности выполнения (циклических конструкций и условных переходов). Для упрощения модели примем, что структура модуля линейна, то есть в нем отсутствуют циклы и ветвления (либо они развернуты в линейную структуру, как это делает оптимизирующий компилятор), а вероятность возникновения ошибки в любой строке кода модуля одинакова и составляет p. Оценка этой вероятности может быть получена либо экспертным образом, либо обработкой статистических данных о прогонах модулей, написанных программистом, с подсчетом относительных частот ошибок. В любом случае, чем квалифицированнее программист, тем ниже эта вероятность.

В результате исполнения линейного модуля из N строк, каждая из которых с вероятностью p содержит дефект, может возникнуть одно из $N+1$ событий: «исполнение модуля на i-ой строке завершилось ошибкой», $i = 1, \ldots, N$, «исполнение модуля прошло без ошибок». Тогда неопределенность относительно результата исполнения модуля может быть измерена показателем информационной энтропии [4, 5]:

$$H = -\sum_{i=1}^{N+1} q_i \log q_i \,. \qquad (1)$$

Здесь q_i – вероятность события, возникающего при запуске модуля. В силу линейности структуры модуля, а также предположения о равенстве вероятностей возникновения ошибки в каждой из строк программного кода, вероятности q_i образуют последовательность: $p, p(1-p), p(1-p)^2, \ldots, p(1-p)^{N-1}, (1-p)^N$.

Таким образом, формула (1) может быть приведена к виду (2):

$$H(p,N) = \\ = -\left(\sum_{i=0}^{N-1} p(1-p)^i \log p(1-p)^i + (1-p)^N \log (1-p)^N \right) \qquad (2)$$

Воспользовавшись свойствами логарифма и суммы, а также тем фактом, что полученные последовательности вероятностей образуют геометрические прогрессии, можно избавиться от суммирования в формуле (2) и получить общее выражение для вычисления энтропии, зависящее

только от вероятности ошибки p и числа строк, потенциально содержащих дефекты, приводящие к ошибке:

$$H(p,N)=\frac{(1-p)^N-1}{p}\big((1-p)\log(1-p)+p\log p\big) \qquad (3)$$

Полученная формула (3) позволяет измерить величину неопределенности относительно результата выполнения модуля, содержащего N строк, в каждой из которых с вероятностью p может произойти ошибка. Величина $H(p, N)$ достигает нуля при $N=0$, то есть при отсутствии строк, в которых имеется вероятность возникновения ошибки.

2. Далее предположим, что код модуля выполняется в среде, предоставляемой системой тестирования. При этом имеется набор модульных тестов, покрывающих k из N строк модуля. Успешное выполнение очередной строки, покрытой модульным тестом, гарантирует, что эта строка не содержит дефекта, а провал теста сигнализирует об обнаруженном дефекте, исправление которого подтверждается повторным успешным прохождением теста. Таким образом, каждый пройденный тест уменьшает неопределенность и увеличивает уверенность разработчика относительно работоспособности модуля. Тогда энтропия относительно работоспособности модуля, содержащего N строк, из которых k покрыты тестами, может быть измерена величиной $H(p, N-k)$ по формуле (3), а *эффективность* модульного тестирования предлагается измерять следующей величиной:

$$E(N,k;p)=1-\frac{H(p,N-k)}{H(p,N)} \qquad (4)$$

Величина (4) равна 0 при $k=0$ и достигает единицы при $k=N$, т. е. при полном покрытии строк модуля тестами. Заметим, что уровень эффективности тестирования модуля зависит только от чисел N, k и вероятности p, и не зависит от выбранного основания логарифма:

$$E(N,k;p)=1-\frac{H(p,N-k)}{H(p,N)}=1-\frac{(1-p)^{N-k}-1}{(1-p)^N-1} \qquad (5)$$

Покажем, что этот показатель сходится к известной метрике тестового покрытия при $p\to 0$. Воспользовавшись эквивалентностью бесконечно малых $((1-p)^m-1)\sim mp$, получаем:

$$E(N,k;p)=1-\frac{(1-p)^{N-k}-1}{(1-p)^N-1}\underset{p\to 0}{\to}1-\frac{N-k}{N}=\frac{k}{N} \qquad (6)$$

Отношение k/N линейно измеряет уровень тестового покрытия через отношение числа строк кода, покрытых тестами, к общему числу строк. В отличие от него, предлагаемый показатель эффективности является нелинейным и зависит от вероятности p, которая косвенно соответствует уровню квалификации программиста. Очевидно, что для программного кода, написанного программистами различной квалификации, может потребоваться набор тестов, покрывающий различное число строк для достижения одного и того же уровня эффективности тестирования модуля.

Рассмотрим пример. Известно [1], что достаточным для практических применений считается покрытие тестами 75% строк программного кода. Предположим, что тестируемый модуль имеет $N=1000$ строк. Тогда отношение k/N достигает величины 0,75 при покрытии тестами $k=750$ строк.

Вычислим эффективность тестирования по формуле (5) в предположении, что $p=0{,}001$ и $p=0{,}005$, и получим значения эффективности $E(1000, 750; 0{,}001) = 0{,}65$ и $E(1000, 750; 0{,}005) = 0{,}28$ соответственно.

Хорошо видно, что увеличение вероятности ошибки в строке снижает значение эффективности тестирования при сохранении неизменным числа покрытых строк. Чтобы в указанных условиях достичь значения показателя эффективности $E\ge 0{,}75$, необходимо в первом случае покрыть тестами не менее $k_1=830$ строк, а во втором случае – уже не менее $k_2=945$ строк кода проверяемого модуля.

Подводя итоги, заметим, что тема выбора подходящих метрических показателей является весьма обсуждаемой среди специалистов в области качества программного обеспечения. Разнообразные подходы, применяемые при измерении метрик с целью оценивания качества, рассматриваются, например, в [11]. Вопросам получения оценки качества программного продукта, интегрирующей на основе формулы Байеса неточные результаты измерений и субъективные результаты экспертиз, посвящены опубликованные ранее работы авторов [12 – 14]. В этой связи необходимо отметить, что полученное авторами и представленное в настоящем докладе в виде формулы (5), выражение для вычисления эффективности тестирования вполне может использоваться как одна из метрик важнейшей характеристики качества программного обеспечения – надежности. При этом преимуществом предлагаемого метрического показателя по сравнению с такой известной метрикой качества тестирования, как тестовое покрытие строк кода, является возможность учета уровня квалификации программиста-разработчика.

IV. Особенности разработки в стиле TDD как информационного процесса

Стандартная разработка программного модуля с использованием тестирования выполняется по спирали «написание кода» – «прогон» – «отладка» – «выпуск», в которой этапы «прогон» – «отладка» повторяются циклично до тех пор, пока все предусмотренные тестовые прогоны не завершатся успешно, без аварийного останова

или отличия результатов работы от эталонных (ожидаемых) результатов.

Разработка программного кода, управляемая тестированием, предполагает создание автоматизированных модульных тестов, определяющих требования к коду до написания самого кода. При использовании TDD как техники разработки кода программистом циклически повторяется следующий набор действий [9, 10]:

- сначала пишется тест, покрывающий требуемое изменение функциональности;
- затем, в случае провала автоматизированного теста, пишется код (не обязательно идеальный), позволяющий пройти тест;
- далее выполняется рефакторинг нового кода (удаление дублирования, улучшение стиля, приведение к действующим для данного проекта или организации стандартам).

Техника TDD повышает качество программного кода: код пишется быстрее (сокращается время, затрачиваемое на отладку), выглядит чище (постоянно выполняется рефакторинг) и является более устойчивым (количество дефектов в коде уменьшается за счет большого объема тестирования).

Рассматривая разработку в стиле TDD в интересующем нас контексте, т. е. как информационный процесс, направленный на уменьшение неопределенности по мере выполнения модульных тестов, можно констатировать следующее: сама логика применяемой техники программирования теоретически обеспечивает максимально возможный уровень эффективности тестирования за счет 100%-го покрытия строк кода тестами.

V. Заключение

В докладе тестирование понимается как информационный процесс, направленный на преодоление неопределенности относительно работоспособности кода программного модуля. Каждый пройденный (с возможным последующим исправлением ошибки и повторением запуска) тест уменьшает неопределенность и увеличивает уверенность разработчика в качестве кода.

Такая интерпретация позволяет предложить вероятностную модель выполнения кода как в реальной среде в составе исполняемой программы, без предварительных тестовых испытаний, так и в среде, формируемой и предоставляемой системой тестирования. Модель используется для получения соответствующих обоим случаям оценок энтропии как меры неопределенности. На этой основе авторами получено выражение для вычисления эффективности модульного тестирования, которое может использоваться как одна из метрик важнейшей характеристики качества программного обеспечения − надежности.

При оценивании качества тестирования широко применяется такая метрика, как тестовое покрытие строк кода. Преимуществом метрического показателя, предлагаемого авторами, по сравнению с этой метрикой является возможность учета уровня квалификации программиста-разработчика, который отражается в используемой при оценивании эффективности тестирования величине вероятности обнаружения дефекта в строке программного кода.

Список литературы

[1] Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2008. 368 с.

[2] Кожомбердиева Г. И. Оценка качества программного обеспечения: учеб. пособие. СПб.: ПГУПС, 2010. 44 с.

[3] Хомоненко А. Д., Кожомбердиева Г. И. Введение в CMMI – комплексную модель зрелости процессов разработки программного обеспечения: учеб. пособие. СПб.: ПГУПС, 2008. 34 с.

[4] Вентцель Е. С. Теория вероятностей. М.: Физматгиз, 1962. 564 с.

[5] Денисов В. В. Основы энтропийно-информационного анализа: учеб. пособие. СПб.: ПГУПС, 2004. 40 с.

[6] Кожомбердиева Г. И., Сухоногов А. М., Протопопов Д. А. Использование средств тестирования JUnit при разработке Java-приложений в среде Oracle JDeveloper: метод. указания. СПб.: ПГУПС, 2014. 33 с.

[7] Кожомбердиева Г. И., Сухоногов А. М., Протопопов Д. А. О применении средств автоматизации модульного тестирования для повышения качества программного кода информационных систем // Интеллектуальные системы на транспорте: материалы IV междунар. научно-практич. конф. «ИнтеллектТранс-2014», Санкт-Петербург, 3-4 апреля 2014 г. СПб.: ПГУПС, 2014. С. 525–530.

[8] Official website JUnit [Электронный ресурс]. – URL: https://junit.org/junit5/ (дата обращения: 02.04.2018)

[9] Бек К. Экстремальное программирование: разработка через тестирование. Библиотека программиста; [пер. с англ.]. СПб.: Питер, 2003. 224 с.

[10] Чурсин П. О. Исследование возможностей техники экстремального программирования TDD (Test Driven Development) при разработке кода Java с использованием фреймворка JUnit /науч. рук. Кожомбердиева Г.И. // Транспорт: проблемы, идеи, перспективы (Неделя науки - 2014): сб. трудов LXXIV всероссийской науч.-техн. конф. студентов, аспирантов и молодых ученых, Санкт-Петербург, 21-25 апреля 2014 г. СПб.: ФГБОУ ВПО ПГУПС, 2014. С. 359–363.

[11] Черников Б. В., Поклонов Б. Е. Оценка качества программного обеспечения: Практикум: учеб. пособие. М.: ИД «ФОРУМ»; ИНФРА-М, 2012. 400 с.

[12] Кожомбердиева Г. И., Бураков Д. П. Об использовании формулы Байеса в задачах оценивания качества // Международная конференция по мягким вычислениям и измерениям: Сб. докл. XX Междунар. конф. SCM’2017, Санкт-Петербург, 24–26 мая 2017 г. СПб: СПбГЭТУ «ЛЭТИ», 2017. Т. 1. С. 31-34.

[13] Бураков Д. П., Кожомбердиева Г. И. Особенности применения байесовского подхода при оценивании качества программных продуктов // Международная конференция по мягким вычислениям и измерениям: Сб. докл. XX Междунар. конф. SCM’2017, Санкт-Петербург, 24–26 мая 2017 г. СПб: СПбГЭТУ «ЛЭТИ», 2017. Т. 1. С. 35-38.

[14] Кожомбердиева Г. И., Бураков Д. П. Получение интегральной оценки качества программного обеспечения на основе формулы Байеса // Транспортные интеллектуальные системы: Сб. материалов I междунар. науч.-практич. конф. «Транспортные интеллектуальные системы – 2017» (TIS-2017), Санкт-Петербург, 16–17 февраля 2017 г. СПб: ФГБОУ ВО ПГУПС, 2017. С. 209–220.