УДК 004.8


А.М. Лужинский, С.Ю. Ржеуцкая
Вологодский государственный университет

ПОДХОДЫ К РЕГРЕССИОННОМУ ТЕСТИРОВАНИЮ В ПРОГРАММНЫХ РАЗРАБОТКАХ НА ОСНОВЕ МЕТОДОЛОГИИ SCRUM

В статье анализируются существующие подходы к регрессионному тестированию в программных проектах, выполняемых с использованием методологии Scrum. В дополнение к имеющимся подходам представлена вероятностная модель взаимодействия между модулями в процессе функционирования программного продукта, использование которой позволит сократить сроки и повысить качество регрессионного тестирования.

Методология Scrum, регрессионное тестирование, тестирование на основе моделей, взаимодействие между модулями.

Большая часть ИТ-компаний в процессе разработки программного обеспечения использует гибкие методологии (Agile). Одной из наиболее известных и востребованных гибких методологий является Scrum, которая регламентирует основные принципы организации процесса разработки, позволяющие в короткие по времени итерации (спринты) предоставлять конечному пользователю работоспособное программное обеспечение, дополненное какими-либо новыми функциональными возможностями. С помощью данной методологии можно постепенно наращивать функционал программного продукта в тесной коммуникации между заказчиком и участниками проекта.

В методологии Scrum важная роль отводится регрессионному тестированию, которое помогает командам убедиться, что новые функциональные возможности, добавленные в очередном спринте, не нарушили уже существующую функциональность [1, 4].

В данной статье анализируются существующие подходы к регрессионному тестированию применительно к Scrum-проектам и предлагается способ тестирования, основанный на вероятностной анализе взаимосвязи между программными единицами в процессе функционирования программного продукта.

При проведении регрессионного тестирования в Scrum-проекте важно как можно более точно выделить программные модули, которые затронули изменения в коде в очередном спринте, а также те модули, которые могут быть затронуты изменениями при наращивании функционала в следующих спринтах. В противном случае придется постоянно тестировать весь код, что более затратно по времени. Также важно выбрать проверки, которые можно автоматизировать. При этом не пренебрегать и так называемым исследовательским тестированием, которое выполняется вручную и помогает лучше понять логику построения и функционирования программного продукта.

Но даже при успешном выполнении всех перечисленных рекомендаций Scrum-команды могут столкнуться с рядом сложностей. Основная из них – возрастающий от спринта к спринту объем регрессии [3]. На крупных проектах с каждым новым спринтом объем регрессионного тестирования будет существенно увеличиваться. Чтобы уложиться в сжатые сроки тестирования в пределах короткого спринта, важно пересматривать тест-кейсы, добавлять новые и удалять устаревшие. В этом процессе требуется непрерывная коммуникация между бизнес-аналитиками, разработчиками и тестировщиками, входящими в проектную команду.

Основой для организации продуктивного взаимодействия всех участников команды проекта может стать тестирование на основе моделей, представляющих структурные и поведенческие аспекты разрабатываемого программного продукта [2]. В качестве широко известных примеров моделей можно назвать диаграммы универсального языка моделирования UML.

В рамках данной статьи представим подход авторов, основанный на математическом моделировании взаимосвязи между программными единицами (модулями) внутри программного продукта. Цель такого моделирования – выделение наиболее значимых программных модулей проекта, которые активнее других используются при реализации различного функционала, следовательно, оказывают наибольшее влияние на функционирование всего продукта в целом. При регрессионном тестировании любые изменения в модулях, выделенных в качестве значимых, должны тестироваться с повышенным вниманием и полным покрытием тестами, желательно автоматизированными. Важно подчеркнуть, что предлагаемый способ моделирования является полезным дополнением к уже существующим подходам к тестированию на основе моделей с целью повышения их эффективности.

Сначала введем несколько определений. Под программной единицей (модулем) будем подразумевать любую подпрограмму, оформленную в строгом соответствии с правилами используемого в проекте языка программирования. Это может быть функция, процедура или метод при объектно-ориентированном подходе к программированию. Тестирование программного кода модуля заключается в проверке корректности работы модуля на различных наборах тестовых данных (выполнении тест-кейсов).

В любом программном модуле можно использовать две различных группы данных:

1. Данные, которые могут использоваться как в самом модуле, так и во внешнем окружении. В эту группу входят параметры подпрограммы, результат, возвращаемый функцией, глобальные переменные или поля класса, значения которых могут измениться при работе модуля.

2. Локальные данные, используемые только внутри модуля, скрытые для внешнего окружения.

Большинство программных модулей содержит в своем коде вызовы других модулей. Это обстоятельство существенно усложняет процесс локализации ошибок в модуле, поскольку заранее не известно, произошла ли ошибка в вызываемом модуле или ему были переданы на вход неправильные исходные данные. Для упрощения процедуры локализации ошибок в модуле стоит ввести контроль для вызываемых модулей: необходимо проверять правильность входных и соответствующих им выходных данных. Такой подход позволит сократить объем анализируемого кода, а также более точно указать место возникновения ошибки.

Поскольку в современном программном обеспечении количество модулей очень велико, реализовать для них всех модульные тесты за разумное время, ограниченное временем спринта, не представляется возможным. В связи с этим и возникает актуальная проблема определения наиболее значимых с точки зрения регрессионного тестирования модулей. Выполнение первоначального и более тщательного тестирования наиболее значимых модулей позволит сократить время тестирования, повысив общее качество тестируемой системы. Таким образом, существует необходимость создания алгоритма выделения наиболее значимых модулей.

Для создания такого алгоритма сначала представим математическую модель взаимосвязи между модулями в процессе функционирования программного продукта. Очевидно, что этот процесс имеет вероятностную природу, поэтому воспользуемся теорией вероятностей.

Рассмотрим две программных единицы (модуля) M_1 и M_2, при этом модуль M_2 (вызывающий) содержит в своем коде вызов модуля M_1 (назовем его вызываемым). Пусть вызываемый модуль возвращает *n* выходных данных, а вызывающий модуль использует *m* из них. Предположим, что вызываемый модуль содержит ошибку, в результате которой будут получены неверные выходные данные. Найдем вероятность того, что вызывающий модуль будет использовать ошибочные данные после окончания работы вызываемого модуля. При этом допустим, что вероятности получения ошибки в любом из *n* выходных данных вызываемого модуля одинаковы.

Будем рассматривать такие события:

A – «Вызывающий модуль будет использовать ошибочные выходные данные, полученные при работе вызываемого модуля».

B – «Вызываемый модуль обязательно содержит ошибку» (иными словами, вероятность $P(B) = 1$).

Введем гипотезы H_i – «Количество ошибочных результатов в вызываемом модуле равно *i*». Поскольку вероятности получения ошибки в любом результате одинаковы, то $P(H_i) = (P(H_1))^i$. На основании формулировки гипотез можно получить выражение:

$$\sum_{i=1}^{n} P(H_i) = 1 \quad \Rightarrow \quad \sum_{i=1}^{n} P(H_1)^i = 1, \qquad (1)$$

тогда вероятность $P(H_1)$ – корень уравнения $x + x^2 + \cdots + x^n = 1$.

В таблице приведены значения вероятностей $P(H_i)$ для *n* от 1 до 5, вычисленные на основе соотношения (1).

Таблица

Значения вероятностей $P(H_i)$

n = 1	$P(H_1) = 1$
n = 2	$P(H_1) = 0{,}618$; $P(H_2) = 0{,}382$
n = 3	$P(H_1) = 0{,}5437$; $P(H_2) = 0{,}2956$; $P(H_3) = 0{,}1607$
n = 4	$P(H_1) = 0{,}5188$; $P(H_2) = 0{,}2692$; $P(H_3) = 0{,}1396$; $P(H_4) = 0{,}0724$
n = 5	$P(H_1) = 0{,}5087$; $P(H_2) = 0{,}2588$; $P(H_3) = 0{,}1316$; $P(H_4) = 0{,}067$; $P(H_5) = 0{,}0339$

Используя формулу полной вероятности, получаем:

$$P(A \mid B) = \sum_{i=1}^{n} \left(P(A \mid H_i) \cdot P(H_i) \right). \qquad (2)$$

В формуле (2) выражение условной вероятности в зависимости от принятых гипотез H_i можно оценить следующим образом:

$$P(A \mid H_i) = \begin{cases} 1, i > n - m \\ \dfrac{m \cdot i}{n}, i \le n - m \end{cases}. \qquad (3)$$

Таким образом, можно получить выражение для условной вероятности события A в зависимости от события B:

$$P(A \mid B) = \sum_{i=1}^{n-m} \left(\frac{m \cdot i}{n} \right) + \sum_{i=n-m+1}^{n} \left(P(H_1)^i \right). \qquad (4)$$

Полученное нами выражение (4) позволяет ввести новую метрику, которая показывает, насколько велико влияние возможных ошибок в каждом конкретном модуле M на функционирование программного продукта:

$$\mu(M) = \sum_{j} P(A_j \mid B), \qquad (5)$$

где событие A_j формулируется так – «Модуль M_j будет использовать ошибочные данные, полученные в результате работы модуля M»; событие B – «Модуль M обязательно содержит ошибку (вероятность ошибки равна единице)».

Однако такая метрика еще не полностью характеризует значимость модуля и важность его первоочередного полноценного тестирования, поскольку не учитывает сложность конкретного программного кода

и, следовательно, вероятность наличия ошибки в программном модуле.

Например, короткая, но достаточно часто используемая в сторонних модулях функция, и более объемная функция, но используемая намного реже, могут иметь одинаковые метрики, подсчитанные по формуле (5). Но для короткой функции вероятность того, что в ней нет ошибки, намного выше. В то время как вторая функция представляется более «опасной» с точки зрения возможности наличия в ней ошибок.

Таким образом, для определения значимости модуля в процессе регрессионного тестирования следует использовать также дополнительный параметр, характеризующий сложность модуля. Существуют различные метрики, позволяющие оценить сложность программного кода, некоторые из них представлены в [1].

Пусть k – коэффициент, характеризующий сложность программного кода модуля, подсчитанный по одной из существующих метрик (выбор метрики – тема отдельного обсуждения). Тогда можно представить окончательную формулу для расчета значимости модуля с точки зрения процесса регрессионного тестирования программного продукта:

$$\mu(M) = k \cdot \sum_{j} P(A_j \mid B). \qquad (6)$$

Опираясь на полученную метрику (6), можно предложить алгоритм ранжирования программных модулей по степени их значимости в процессе регрессионного тестирования в Scrum-проектах. Он будет состоять в подсчете метрики значимости для каждого модуля по приведенной выше формуле с последующей сортировкой результатов по убыванию. Полученные результаты позволят выделить наиболее значимые модули, для которых целесообразно подготовить автоматизированные тесты, а также отсечь маловажные модули, проверку которых проще выполнить вручную. Уровень значимости программного модуля, ниже которого модули можно признать мало значимыми для программного продукта, определяется в каждом проекте на основе результатов анализа.

Результаты экспериментов по применению предложенного алгоритма в процессе регрессионного тестирования в Scrum-проектах показывают, что лежащая в его основе модель взаимодействия между модулями позволяет с высокой степенью достоверности определять значимость программных модулей и выделять те из них, которые таят в себе потенциальную опасность и должны быть подвергнуты первоочередному, особо тщательному тестированию. Эксперименты в данном направлении продолжаются. В конечном итоге, предлагаемое расширение традиционных подходов к регрессионному тестированию в Scrum-проектах позволит повысить качество протестированного программного кода и уложиться в разумные сроки тестирования. Качественное регрессионное тестирование позволит компании, работающей по методологии Scrum, поднять свою конкурентоспособность на рынке программных разработок.

Литература

1. Звездин, С. В. Проблемы измерения качества программного кода / С. В. Звездин // Вестник Южно-Уральского государственного университета. – 2010. – № 2. – С. 82–86.

2. Кулямин, В. В. Архитектура среды тестирования на основе моделей, построенная на базе компонентных технологий / В. В. Кулямин // Труды института системного программирования. – Москва, 2010. – Т. 18. – С. 9–43.

3. Литвинова, О. Оптимизация регрессионного тестирования в гибкой разработке / О. Литвинова. – Текст электронный // Корпорация ScienceSoft USA [сайт]. – URL: https://www.scnsoft.com/software-testing/regression-testing-in-agile-development (дата обращения: 10.02.2022).

4. Сергушичева, А. П. Технологии разработки программного обеспечения : учебное пособие / А. П. Сергушичева. – Вологда : ВоГУ, 2019. – 92 с.

A.M. Luzhinsky, S.Yu. Rzheutskaya
Vologda State University

APPROACHES TO REGRESSION TESTING IN SOFTWARE DEVELOPMENT BASED ON SCRUM METHODOLOGY

The article analyzes the existing approaches to regression testing in software projects performed with the use of Scrum methodology. In addition to the existing approaches, a probabilistic model of interaction between modules in the process of functioning of the software product is presented; the use of which will reduce the time and improve the quality of regression testing.

Scrum methodology, regression testing, model-based testing, interaction between modules.