УДК 004.415.532 DOI: 10.18698/2541-8009-2018-12-413

МЕТОДИКИ ТЕСТИРОВАНИЯ ПРОДУКТА ОТ ЕГО РАЗРАБОТКИ ДО ВЫПУСКА

Н.Ю. Кисарева kisareva@yandex.ru
SPIN-код: 8743-9202

А.Т. Левинский adam.levinskiy@yandex.ru
SPIN-код: 2301-6960

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Проведено исследование современных стратегий тестирования, выполнена классификация по уровням возможного тестирования продукта с момента его разработки до момента выпуска, описаны методики, присущие каждому уровню, проанализированы недостатки и преимущества каждого из них. Объектом исследования данной статьи являются методики тестирования программного обеспечения продукта, сопровождающие его от начала разработки до выпуска на рынок. Цель работы — исследование и модернизация современных стратегий и тактик тестирования, классификация методов тестирования, анализ их преимуществ и недостатков, а также вывод стратегии, позволяющей провести максимально полное тестирование в сжатые сроки и исключить возможные ошибки в тестируемом программном обеспечении.

Ключевые слова

Тестирование, программное обеспечение, методика, разработка, модуль, модульное тестирование, интеграция, альфа-тестирование

Поступила в редакцию 06.11.2018

Введение. С каждым годом уровень сложности и важности программного обеспечения существенно возрастает, при этом тестирование программного обеспечения становится неотъемлемой и значительной частью его создания. В настоящее время реальные программные продукты нередко разрабатываются в очень короткие, сжатые сроки и при достаточно ограниченном бюджете всего проекта в целом. Программирование сегодня — это уже не что-то недостижимое, не что-то непонятное, а вполне привычное дело для многих специалистов. К сожалению, в такой спешке разработчики нередко игнорируют необходимость обеспечения надлежащего качества, тем самым подвергая риску качественное и полезное использование своего продукта. Именно эти функции и выполняет тестирование.

Тестирование программного обеспечения — это методы и средства оценки разрабатываемого программного продукта, направленные на проверку его возможностей, способностей, соответствия ожидаемым результатам, процесс исследования программного продукта, имеющий своей целью проверку соответ-

ствия реального поведения программы ее ожидаемому поведению на конечном наборе тестов, выбранных определенным образом [1]. Тестирование программного продукта — одна из самых важных и неотъемлемых частей разработки программного обеспечения.

Основным аспектом, доказывающим актуальность тестирования в процессе разработки программного обеспечения, является минимизация непредвиденных затрат как разработчика, так и потребителя продукта. Эти затраты связаны с нарушением процесса разработки и применения программного продукта, вызванным необходимостью устранения найденных в программе ошибок — дефектов. Дефекты, обнаруженные и устраненные уже на ранней стадии разработки, обходятся разработчику и клиенту в десятки и даже сотни раз дешевле, чем такие же, но вскрывшиеся уже в период коммерческого использования продукта.

Тестирование полезно еще и тем, что позволяет собрать информацию об уже совершенных в процессе разработки ошибках. Своевременное обеспечение подобной информацией разработчиков и руководителей проектов существенно снижает риск возникновения повторных дефектов, что в итоге положительно сказывается на качестве программного продукта.

По мере выпуска продукта он проходит различные уровни тестирования. На каждом из этих уровней осуществляется отсечение ошибок, присущих только этому уровню. Таким образом, к моменту выхода продукта большинство возможных ошибок сводится к минимуму при проведении правильного, корректного тестирования.

Невозможность полного покрытия тестами. Тестирование программного обеспечения — это, по сути, процесс поиска ошибок в реализации программы [2]. Естественно, хотелось бы организовать тестирование таким образом, чтобы выявить все возможные ошибки программы. Можно утверждать, что правильность и абсолютную свободу от ошибок программы гарантируют:

– подготовленные все возможные наборы входных данных (в том числе и некорректные);

– выполнение программы при всех возможных перестановках и сочетаниях этих входных данных;

– анализ всех выходных данных и установка факта, что каждый тестовый выходной набор правильный.

Видно, что для реализации всех вышеприведенных пунктов требуются колоссальные затраты времени, финансовые и человеческие ресурсы, ведь даже для небольших программ с парой входных параметров число тестовых случаев может исчисляться миллионами комбинаций. Это доказывает, что исчерпывающее тестирование программного обеспечения невозможно. Следовательно, невозможно выявить все ошибки в реализации программы. Именно поэтому интуитивный процесс тестирования не приносит, как правило, большой пользы, тем более что в условиях жесткого ограничения времени и средств на разработку выбор маленького подмножества тестовых ситуаций часто является вы-

нужденной необходимостью. Полный выбор необходимо делать не спонтанно, а на основании определенной стратегии, которой следует придерживаться при тестировании продукта. Это доказывает, что исследование современных методов, позволяющих проводить тестирование систематически, а не интуитивно, является актуальной проблемой.

Модульное тестирование. Первый уровень — это модульное тестирование. Также его называют единичным тестированием или тестированием компонентов. Из названия понятно, что данный уровень заключается в тестировании единицы программного кода либо набора из одного или нескольких программных модулей, логически выполняющих одну функцию, вместе с соответствующими управляющими данными, процедурами использования и обработки [3]. Идея этого метода — в написании тестов для каждого логически выделяемого класса, функции или метода, а задача — быстро, систематизировано проверить, не привело ли очередное внесение изменений в код к появлению ошибок в тех частях кода, которые ранее уже подвергались тестированию [4].

У данного уровня есть ряд достоинств и недостатков.

К его достоинствам можно отнести следующие.

• *Возможность проведения рефакторинга*. Разработчики могут совершенно спокойно проводить изменение кода и быть уверенными в том, что данный класс работает правильно.

• *Упрощение при последующем интеграционном тестировании* (про интеграционное тестирование см. ниже). На данном уровне тестирования есть возможность убедиться в корректной работе каждого логически выделяемого модуля, что в дальнейшем может быть использовано для тестирования снизу вверх — сначала отдельные части программы (модульное тестирование), затем проверка групп отдельных модулей (интеграционное тестирование), а после проверка всей программы целиком [5].

• *Документирование кода.* Тесты, проводимые на данном уровне, можно использовать как основу документации тестируемого модуля. Пользователи, которые не будут знать, как использовать данный класс, смогут применять эти тесты в качестве примеров.

• *Минимизация зависимостей в системе*. Поскольку довольно часто классы могут наследовать, расширять другие классы, тестирование одного класса часто оказывается связанным с тестированием других классов. К примеру, во время тестирования разработчик находит зависимость между тестируемым им классом и каким-либо другим. Это ошибка, поскольку на данном уровне тестирования тесты должны проводиться только в рамках данного метода (класса, модуля) и не должны выходить за их границы. В результате разработчик должен абстрагироваться от другого класса и реализовать этот интерфейс, используя так называемые объекты-заглушки, что приведет к менее связанному коду и минимизирует зависимости в системе.

Однако модульное тестирование обладает также рядом недостатков.

• *Невозможность применения методики при присутствии сложного кода.* Чем сложнее код — тем сложнее отследить все возможные исходы программы и отловить все ошибки программы. В основном на данном уровне отслеживаются простейшие случаи.

• *Приблизительность результата*. Довольно сложно написать тесты на каждый модуль, класс, функцию и также сложно определить, насколько верен результат. В основном это не документировано и верность определяется довольно субъективно и примерно.

• *Невозможность проверить код, взаимодействующий с какой-либо системой.* Если код взаимодействует с портами, таймерами или другими нестабильными частями системы, довольно трудно проверить его в изолированном окружении.

• *Ошибки взаимодействия между модулями*. Ошибки, связанные с взаимодействием между классами, модулями, функциями, на данном уровне тестирования обнаружить нельзя. Из этого можно сделать вывод, что не стоит выделять какие-то одни уровни тестирования и пренебрегать другими — все они эффективны только в сочетании друг с другом.

• *Проблемы с объектами-заглушками*. В готовой системе все ее части взаимодействуют друг с другом, а также с другими объектами. На этом же уровне рассматриваются только простейшие, стандартные случаи взаимодействия. Для тестирования выполняют подобные этим другим объектам устройства — объекты-заглушки (mock-объекты), которые делают либо крайне упрощенными, либо рассчитанными на конкретный тест. В реальной ситуации настоящие устройства могут вести себя не так, как вели себя заглушки.

• *Субъективность тестирования*. Поскольку данная методика предполагает написание тестов программистом, ранее разработавшим этот же код, то в некоторых случаях оценка может быть весьма субъективна.

Интеграционное тестирование. После модульного тестирования, как правило, проводят интеграционное тестирование, при котором отдельные программные модули объединяют и тестируют в группе.

При интеграционном тестировании в качестве входных данных использует модули, над которыми ранее было выполнено модульное тестирование, группирует их в более крупные множества, реализует тесты, определенные в плане тестирования для этих множеств и представляет их в качестве выходных данных, входных для последующего уровня тестирования — системного.

Как правило, тестирование этих групп выполняют с использованием тестирования методом «черного ящика» [6].

Перечислим преимущества интеграционного тестирования.

• *Более полное покрытие протестированных частей*. Поскольку тестирование производится после объединения в группы программных модулей, есть возможность исключить тестирование с объектами-заглушками (что было недостатком в модульном тестировании), а соответственно провести тестирование не только в общих, стандартных ситуациях, но и в нестандартных [7].

• *Присутствие систем непрерывной интеграции*. Существуют системы непрерывной интеграции, позволяющие автоматизировать интеграционное тестирование, обнаруживать и устранять ошибки в короткие сроки.

Интеграционное тестирование обладает одним серьезным недостатком.

Если в модульном тестировании понятно, что именно нужно тестировать, то в интеграционном тестировщики сталкиваются с проблемами — нет определенных «правил», по которым можно определить, какие именно модули нужно объединить в группы, и ответственность за логичность и рациональность разделения ложится в основном на самого тестировщика. Часто объединение в группы происходит неправильно, и тогда либо происходит повторение модульного тестирования (слишком много групп), либо большинство важных связей остаются непроверенными (слишком мало групп).

Системное тестирование. После интеграционного тестирования проводят системное тестирование — тестирование, выполняемое на полной, интегрированной системе, тестирование программы в целом [2]. Как правило, системное тестирование не требует знаний о внутреннем устройстве системы, поэтому относится к методам тестирования черного ящика и выполняется не разработчиками программы, а непосредственно тестировщиками.

На данном уровне осуществляют следующие системные тесты:

– функциональное тестирование (тестирование программного обеспечения в целях проверить реализуемость функциональных требований);

– тестирование пользовательского интерфейса (или юзабилити-тестирование — тестирование, целью которого является определение удобства интерфейса программы для конечного пользователя);

– тестирование совместимости (проверка, насколько данная версия программного обеспечения совместима с оборудованием, предыдущими версиями, операционной системой и т. п.);

– тестирование на основе модели (варианты тестирования получают на основе модели, описывающей некоторые аспекты тестируемой системы);

– тестирование безопасности (оценка уязвимости программного обеспечения к различным атакам);

– тестирование производительности (тестирование, целью которого является установление быстродействия системы под определенной нагрузкой; помогает определить и другие характеристики системы, к примеру, ее надежность, масштабируемость);

– регрессионное тестирование (проверка уже протестированных ранее частей исходного кода);

– автоматическое (или автоматизированное) тестирование (тестирование, при котором используются программные средства для выполнения тестов).

Очевидно, что системное тестирование является довольно объемным и требует гораздо больше ресурсов и усилий, чем модульное и интеграционное тестирование. Системное тестирование для большей надежности подразделяют на альфа- и бета-тестирование.

Альфа-тестирование — это тестирование (как правило, ручное) потенциальными пользователями, независимой командой тестировщиков, проводимое непосредственно перед бета-тестированием. Чаще всего альфа-тестирование проводят на ранней стадии разработки, под отладчиком, а найденные ошибки могут быть в дальнейшем переданы тестировщикам для более детального исследования в приближенной для пользования системе. Версия, с которой осуществляют альфа-тестирование, называется альфа-версией.

По завершении тестирования альфа-версии выпускают бета-версию, которая представляет собой уже реально работающую программу с полным функционалом. В роли бета-тестировщиков, как правило, выступают люди, имеющие опыт работы с такими программами или с предыдущей версией этого же программного обеспечения. Бывает и такое, что бета-версию выпускают для того, чтобы получить обратную связь от пользователей, либо в качестве своеобразной рекламы для получения предварительных отзывов.

Перечислим основные преимущества альфа-тестирования.

• *Большое покрытие.* Данные тесты покрывают практически все группы модулей программного обеспечения и сводят к минимуму возможные ошибки в программе.

• *Взаимодействие с пользователем.* Поскольку на данном уровне имеют место быть такие типы тестирования, как тестирование пользовательского интерфейса, бета-тестирование, то конечный продукт, при корректном проведении тестов, максимально приближен к желаниям конечного пользователя, что является одной из главных целей разработки программного обеспечения.

• *Предварительное выявление ошибок в непредвиденных ситуациях.* Тестирование безопасности, тестирование совместимости, тестирование производительности позволяют преждевременно отыскать или предотвратить ошибки, получаемые в нестандартных, стрессовых ситуациях, а следовательно, минимизировать возможные затраты на восполнение убытков «пострадавшим» пользователям.

Присутствуют также определенные недостатки.

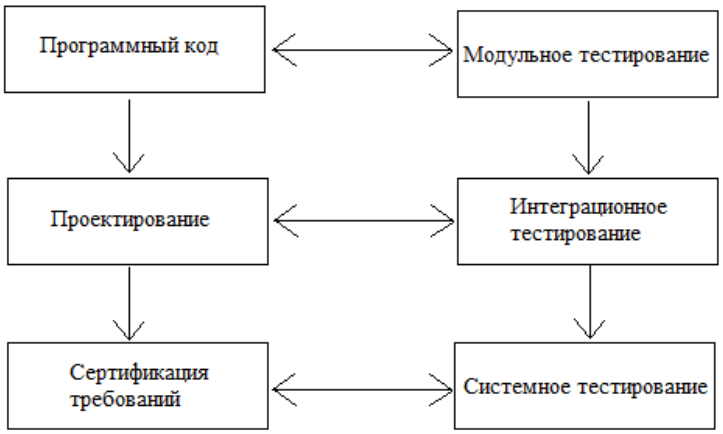
• *Большое покрытие.* Данный пункт является как достоинством, так и недостатком. Не всегда есть возможности, ресурсы и даже время на проведение всех указанных тестов. Длительное тестирование влечет за собой потерю интереса со стороны пользователей и как следствие — потерю клиентов.

• *Возможность создания плохой рекламы.* При выпуске на стадии бета-тестирования не полностью протестированного программного продукта с возможными ошибками велика вероятность распространения негативных отзывов со стороны потенциальных пользователей.

Описанные уровни тестирования и часть проекта, которая проверяется на конкретном уровне, показаны на рисунке.

Тренды развития тестирования программного обеспечения. Мы живем в век информатики, а информационные технологии на месте долго не стоят: меняются роль участников цикла разработки программного обеспечения, появляются новые инструменты, подходы, возникает необходимость приобретения

новых навыков. Хороший тестировщик должен не останавливаться в своем развитии и пристально следить за всеми новинками в индустрии информационных технологий. Эффективность тестирования, конечно, сильно зависит от четко сформулированной, выбранной не наугад стратегии, но данное тестирование не будет столь эффективно при использовании неактуальных и несовременных средств, с помощью которых оно проводится.



Этапы тестирования

Развитие интернета вещей. Количество подключенных к всемирной паутине вещей в мире растет. Сферы их применения расширяются: энергетика, промышленность, транспорт, здравоохранение и т. д. При этом разработка и внедрение «умных» вещей может вестись по инициативе как бизнеса, так и государства.

Внедрение IoT-технологий — сложный процесс, который требует тщательно продуманной стратегии разработки и тестирования [8].

Обладая знаниями и практическим опытом, тестировщики могут провести грамотное тестирование функциональности, производительности, совместимости с различными операционными системами, браузерами и, конечно, безопасности. В основе работы подключенных устройств лежат данные. Важно убедиться в том, что доступ к данным надежно защищен.

Кроме того, тестировщик должен уметь фокусироваться на различных сценариях использования продукта, какими бы невероятными они ни казались. Так что навыки исследовательского тестирования окажутся как никогда кстати.

Использование инструментов open source. Уже сегодня многие компании используют сервисы open source (открытые источники) для управления тестированием и внедрения автоматизации, поддержки принципов Agile и DevOps, управления жизненным циклом дефектов. Это означает, что в ближайшем будущем станут популярны инструменты open source — гибкие, эффективные, с различными областями применения.

Selenium, Appium, JMeter — инструменты с открытым кодом, которые уже давно используются автоматизаторами. Однако посредством общедоступных

инструментов можно выполнять не только автоматизацию тестирования. Так, для тестирования API можно применять инструмент SoapUI, для нагрузочного тестирования — Apache JMeter.

Инструменты с открытым доступом легко подстраиваются под нужды проекта. Богатая функциональность и поддержка со стороны сообщества снижают стоимость разработки и ускоряют выход продукта на рынок. Именно поэтому заказчики часто требуют, чтобы команда по тестированию работала с инструментами open source. Тестировщикам нужно всегда быть готовыми изучать их. Понятие Big Data у всех на слуху уже не первый год. Но далеко не все тестировщики точно знают, что означает это понятие, хотя и должны, ведь тестирование напрямую связано с данными.

Так что же такое «большие данные»? Определений можно найти много, но все они сводятся к тому, что Big Data связано с объемом и обработкой. Раньше мы тестировали мегабайты и гигабайты, а теперь все чаще речь идет о терабайтах и петабайтах. Однако дело не только в количестве нулей. Большие данные связаны с так называемыми четырьмя V: *volume* (объем), *velocity* (скорость генерации новых данных), *variety* (разнообразие данных) и *veracity* (достоверность данных). Проблема больших данных состоит в том, что они всегда плохо структурированы. Большинство этих данных получено из социальных сетей. Это звонки, сообщения, поисковые запросы, картинки, PDF-файлы и прочая информация.

Тестирование больших данных предполагает функциональные и нефункциональные проверки. Функциональное тестирование включает проверку качества данных и их обработку. К нефункциональным видам тестирования будут относиться нагрузочное тестирование, объемное тестирование. Однако каждый вид тестирования требует от тестировщика знания системы и умения исследовать продукт. При тестировании больших данных часто внедряют автоматизацию.

Гибкая методология разработки Agile и практики DevOps (тесное взаимодействие разработчиков со специалистами по операционной деятельности) определяет место тестировщиков и влияет на взаимодействие внутри команды тестирования [9].

Принципы Agile и DevOps будут применяться все шире, и тестировщики должны разбираться в них и уметь к ним адаптироваться. Так, растущая популярность DevOps свидетельствует о размытии границ. Все меньше становится разграничений между разработкой и этапом эксплуатации программы. Что это значит для тестировщиков? Им придется больше тестировать на этапе эксплуатации, чем на тестовых стендах. Тестировщики обязаны учиться взаимодействовать с командой разработки, чтобы добиться успеха.

Успешное участие в Agile-проектах также требует от тестировщиков умение вести коммуникацию с разработкой и бизнесом, быстро реагировать на изменения требований, определять покрытие для тестирования в условиях сжатых сроков.

Множество технологий, которые находятся на стадии развития, сильно повлияют на тестирование. Особенно это относится к искусственному интеллекту и машинному обучению. Многие полагают, что они станут неотъемлемым эле-

ментом тестирования. Уже сейчас количество систем, интегрирующих машинное обучение, вызывает новые сложности при тестировании, и в будущем это влияние увеличится.

Выводы. В результате анализа современных методов тестирования программного обеспечения установлено, что не существует универсального средства избавления от ошибок программного обеспечения. Каждый из рассмотренных уровней и методов на них ориентирован на выявление определенного класса дефектов.

Тестирование играет немаловажную роль в производстве программного обеспечения, однако в настоящее время еще не разработаны универсальные подходы в тестировании программного обеспечения. В работе были рассмотрены основные стратегии, применяемые при тестировании программного обеспечения. Существуют различные уровни тестирования, которые сопровождают продукт от стадии его разработки до конечного представления пользователям: модульное тестирование (тестирование единицы программного кода), интеграционное тестирование (тестирование групп модулей, связанных между собой) и системное тестирование (тестирование полной программы). Установлено, что исчерпывающее тестирование невозможно вследствие большого числа сочетаний и перестановок всевозможных входных данных. Именно поэтому из всего многообразия входных наборов данных в тестовый набор приходится выделять некоторое мизерное подмножество. Чтобы добиться наилучшего качества и выявить максимальное число дефектов, тестирование необходимо проводить, опираясь на существующие методы. Также в ходе исследования было установлено, что каждый из существующих уровней и методов, присущих этому уровню, ориентирован на выявление определенного типа ошибок. Все уровни тесно связаны между собой и пренебрежение методиками тестирования на одном из уровней может повлечь за собой серьезные последствия, поэтому максимальной эффективности тестирования можно достичь только при совокупном проведении методов тестирования на всех уровнях — для обеспечения приемлемого качества программного обеспечения целесообразно применять комплексные подходы к тестированию, внедряя работы по тестированию в различные этапы технологического цикла разработки программного обеспечения.

Но полностью продуманная стратегия не всегда означает эффективность. Для достижения максимальной эффективности при тестировании следует применять современные, актуальные средства.

Литература

[1] Майерс Г., Баджетт Т., Сандлер К. *Искусство тестирования программ*. Москва, Диалектика, 2012, 271 с.

[2] Криспин Л., Грегори Дж. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. Москва, Вильямс, 2010, 464 с.

[3] Канер К., Фолк Д., Нгуен Е.К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. Киев, ДиаСофт, 2001, 544 с.

[4] Калбертсон Р., Браун К., Кобб Г. Быстрое тестирование. Москва, Вильямс, 2002, 384 с.
[5] Синицын С.В., Налютин Н.Ю. Верификация программного обеспечения, Москва, МИФИ, 2006, 157 с.
[6] Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. Санкт-Петербург, Питер, 2004, 321 с.
[7] Bradtke R. ISTQB 100 success secrets – ISTQB foundation certification software testing the ISTQB certified software tester 100 most asked questions. Emereo, 2008, pp. 35–38.
[8] Нильсен Я., Лоранжер Х. Web-дизайн: удобство использования Web-сайтов. Москва, Вильямс, 2007, 368 с.
[9] Berg A.M. Jenkins continuous integration cookbook. Packt, 2015, 360 p.

Кисарева Надежда Юрьевна — студентка магистратуры кафедры «Информационные системы и телекоммуникации», МГТУ им. Н.Э. Баумана, Российская Федерация.

Левинский Адам Тагирович — студент магистратуры кафедры «Информационные системы и телекоммуникации», МГТУ им. Н.Э. Баумана, Российская Федерация.

PRODUCT TESTING TECHNIQUES FROM ITS DEVELOPMENT TO PRODUCTION

N.Yu. Kisareva kisareva@yandex.ru
SPIN-code: 8743-9202

A.T. Levinsky adam.levinskiy@yandex.ru
SPIN-code: 2301-6960

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

In the article the research of modern testing strategies was carried out, it was made the classification according to the levels of possible product testing from the moment of its development to the moment of production, the techniques inherent to each level were described, the advantages and disadvantages of each of them were analyzed. The object of the research of this article are the methods of testing software product, accompanying it from the start of development to release to market. The goal of the work is to study and modernize present-day strategies and tactics of testing, to classify testing methods, to analyze their advantages and disadvantages, and also to derive strategy that allows for the most complete testing in a short time and to eliminate possible errors in the software under test.

Keywords

Testing, software, methodology, development, module, unit testing, integration, alpha-testing

Received 06.11.2018

References

[1] Myers G.J., Sandler C., Badgett T. The art of software testing. Wiley, 2011, 256 p. (Russ. ed.: Iskusstvo testirovaniya programm. Moscow, Dialektika publ., 2012, 271 p.)

[2] Gregory J., Crispin L. Agile testing: a practical guide for testers and agile teams. Addison-Wesley Professional, 2009, 576 p. (Russ. ed.: Gibkoe testirovanie: prakticheskoe rukovodstvo dlya testirovshchikov PO i gibkikh komand. Moscow, Vil'yams publ., 2010, 464 p.)

[3] Kaner C., Falk J.L., Nguyen H.Q. Testing computer software. Van Nostrand Reinhold, 1993, 480 p. (Russ. ed.: Testirovanie programmnogo obespecheniya. Fundamental'nye kontseptsii menedzhmenta biznes-prilozheniy. Kiev, DiaSoft publ., 2001, 544 p.)

[4] Culbertson R., Brown C., Cobb G. Rapid testing. Prentice Hall, 2002, 416 p. (Russ. ed.: Bystroe testirovanie. Moscow, Vil'yams publ., 2002, 384 p.)

[5] Sinitsyn S.V., Nalyutin N.Yu. Verifikatsiya programmnogo obespecheniya [Software verification]. Moscow, MEPhI publ., 2006, 157 p.

[6] Beizer B. Black box testing. Wiley, 1995, 324 p. (Russ. ed.: Testirovanie chernogo yashchika. Tekhnologii funktsional'nogo testirovaniya programmnogo obespecheniya i sistem. Sankt-Peterburg, Piter publ., 2004, 321 p.)

[7] Bradtke R. ISTQB 100 success secrets - ISTQB foundation certification software testing the ISTQB certified software tester 100 most asked questions. Emereo, 2008, pp. 35–38.

[8] Nielsen J., Loranger H. Prioritizing web usability. New Riders, 2006, 432 p. (Russ. ed.: Web-dizayn: udobstvo ispol'zovaniya Web-saytov. Moscow, Vil'yams publ., 2007, 368 p.)
[9] Berg A.M. Jenkins continuous integration cookbook. Packt, 2015, 360 p.

N.Yu. Kisareva — Master's Degree student, Department of Information Systems and Telecommunications, Bauman Moscow State Technical University, Moscow, Russian Federation.

A.T. Levinsky — Master's Degree student, Department of Information Systems and Telecommunications, Bauman Moscow State Technical University, Moscow, Russian Federation.