ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ
INFORMATION TECHNOLOGY, COMPUTER SCIENCE AND MANAGEMENT

УДК 004.05
https://doi.org/10.23947/2687-1653-2024-24-3-255-263

Оригинальное теоретическое исследование

Управление качеством при разработке программного обеспечения

М.Д. Бируля
EPAM Systems Sp z o.o., г. Краков, Республика Польша
✉ martinbirulia@gmail.com

EDN: JBGRGQ

Аннотация

Введение. В научной литературе рассматриваются разные подходы к менеджменту качества в сфере информационных технологий (ИТ). Проработаны вопросы выявления и исправления дефектов, показаны возможности их минимизации. Есть материалы об управлении качеством в сложных технологических процессах. Доказано, что работа с качеством цифровых продуктов требует в числе прочего прояснения вопросов качества кода. При этом нет детального описания управления качеством на каждом этапе жизненного цикла ИТ-продукта, включая тестирование. Отметим, что координация релизов (выпусков) программного обеспечения тесно связана с управлением качеством, однако данный процесс редко или фрагментарно рассматривается в литературе. К тому же не учитывается взаимодействие процессов, поэтому нет комплексного представления об управлении качеством при создании, тестировании и доработке программного обеспечения (ПО). Данное исследование призвано восполнить указанные пробелы. Его цель — представить комплексный подход, связывающий теорию, практику и методы управления качеством ПО.

Материалы и методы. Исследована, проанализирована и отреферирована профильная теоретическая и прикладная литература. Задействован профессиональный опыт автора в управлении качеством ИТ-продуктов. Учтены практики глобальных поставщиков цифровых товаров и услуг. Автор использовал эти материалы и методы для детальной проработки вопросов тестирования ПО и развертывания кода.

Результаты исследования. Сформирована, описана и представлена в виде схемы комплексная модель управления качеством при создании ПО. Выявлены ее взаимосвязи с моделью менеджмента проектов и жизненным циклом продукта, а именно: анализом, дизайном, разработкой, тестированием, развертыванием и поддержкой. Указаны принципы управления качеством на каждой из этих стадий. Систематизированы и представлены в виде схемы процессы и проверки при развертывании кода. Показаны их особенности в трех средах: при разработке, тестировании и производстве.

Обсуждение и заключение. Алгоритм позволяет специалистам по качеству выстроить последовательность действий для исключения в будущем выявленных дефектов, понимания ситуации, когда можно (или нельзя) развертывать код и определения момента, когда следует передать ПО пользователю. Кроме того, предложенная схема может быть базой для автоматизации развертывания кода. Решение позволит сократить время на разработку. Как следствие, продукт быстрее выйдет на рынок, что ускорит окупаемость затрат. Внедрение в производственную практику ИТ-компаний модели, созданной в рамках данной научной работы, предполагает стратегические изменения. Их реализация требует значительных затрат времени и других ресурсов, поэтому общий процесс трансформаций следует разбить на части. Предложенный подход адаптируется под нужды различных организаций и продуктов. Можно работать с отдельными компонентами, чтобы создать оптимальный план для достижения целей по управлению качеством.

Ключевые слова: развертывание кода, управление качеством в ИТ-сфере, жизненный цикл ИТ-продукта, тестирование ИТ-продукта, релиз программного обеспечения

Благодарности. Автор выражает благодарность ведущим организациям в сфере разработки программного обеспечения — EPAM Systems, Amazon и Google за их практики, которые анализировались при подготовке статьи. Особая благодарность компании Project Management Institute за обновление подходов к управлению проектами.

Для цитирования. Бируля М.Д. Управление качеством при разработке программного обеспечения. *Advanced Engineering Research (Rostov-on-Don).* 2024;24(3):255–263. https://doi.org/10.23947/2687-1653-2024-24-3-255-263

Original Theoretical Research

Quality Management in Software Development

Martin D. Birulia
EPAM Systems Sp z o.o., Kraków, Republic of Poland
✉ martinbirulia@gmail.com

Abstract

Introduction. The scientific literature examines various approaches to quality management in information technology (IT). The issues of identifying and correcting defects are worked out, and the possibilities for minimizing them are shown. There are materials on quality management in complex engineering processes. At the same time, there is no detailed description of quality management at each stage of the IT product life cycle, including testing. It should be noted that the coordination of software releases is closely related to quality management, but this process is rarely or fragmentarily considered in the literature. Additionally, the interprocess communication is not taken into account; therefore, there is no comprehensive understanding of quality management in the creation, testing and refinement of software. This study is designed to fill these gaps. The research is aimed at presenting a comprehensive approach that links the theory, practice and methods of software quality management.

Materials and Methods. Theoretical and applied literature on the subject were studied, analyzed, and reviewed. The author's professional background in managing the quality of IT products was used. The practices of global suppliers of digital goods and services were taken into account. The author has used these materials and methods to study in detail the issues of software testing and code deployment.

Results. A comprehensive model of quality management in software development is elaborated, described and presented in the form of a diagram. Its interconnections with the project management model and the product life cycle, namely: analysis, design, development, testing, deployment, and support, are identified. Principles of quality management at each of these stages are specified. The processes and checks during code deployment are systematized and presented in the form of a diagram. Their features are shown in three environments: during development, testing, and production.

Discussion and Conclusion. The algorithm allows quality experts to build the sequence of actions to eliminate detected defects in the future**,** understand the situation when it is possible (or impossible) to deploy code, and determine the moment when the software should be transferred to the user. In addition, the proposed scheme can be the basis for automating code deployment. The solution will reduce development time. As a result, the product will enter the market faster, which will speed up the payback of costs. The implementation of the model created within the framework of this scientific work into the production practice of IT companies presupposes strategic changes. Their implementation requires significant time and other resources; therefore, the overall transformation process should be divided into parts. The proposed approach is adaptable to the needs of various organizations and products. You can work with individual components to create an optimal plan for achieving quality management goals.

Keywords: code deployment, quality management in IT, IT product life cycle, IT product testing, software release

Acknowledgments. The author appreciates the leading software development organizations, such as EPAM Systems, Amazon, and Google, for their practices, which were analyzed under the preparation of this article. Special thanks to Project Management Institute for updating approaches to project management.

For Citation. Birulia MD. Quality Management in Software Development. *Advanced Engineering Research (Rostov-on-Don).* 2024;24(3):255–263. https://doi.org/10.23947/2687-1653-2024-24-3-255-263

Введение. В условиях высокой конкуренции на рынке программного обеспечения (ПО) пользователи ориентируются не только на маркетинговую, но и на потребительскую ценность товара, обращают внимание на удобный интерфейс, оперативность и стабильность работы. Все это следует учитывать производителям и подразделениям софтверных компаний, которые работают с качеством. В открытом доступе достаточно теоретических и прикладных публикаций, посвященных управлению качеством и автоматизации этих процессов. Авторы рассматривают новации в этой сфере, сравнивают их с традиционными практиками.

Работа с дефектами (в частности, их раннее выявление и анализ) позволяет предотвращать ошибки на этапе разработки софта. При таком подходе сокращаются затраты времени, денег и других ресурсов на исправление логических, синтаксических, компиляционных и других ошибок ПО. Улучшается качество продукта, повышается его ценность в плане надежности, легкости обслуживания и экономичности [1]. В [2] описано управление качеством в сложных технологических процессах. Автор показывает, как выявить взаимосвязь между технологией и качеством. По его предположению, существует информационное соответствие между вероятностными моделями технологии и качеством. Из [3] известно, что в сфере информационных технологий (ИТ) увеличение бюджета на 49 % приводит к созданию новых моделей управления качеством. В качестве примера можно назвать стандарты ISO/IEC [4]. В [5] показано, как управление качеством влияет на производственные процессы.

Важная и сложная задача — автоматизация контроля процессов, от которых зависит качество. Ключевая форма контроля — статистическая. Она работает с индикаторами, критичными для качества конечного продукта (от англ. critical to quality (CTQ)). Эти показатели отслеживаются и сравниваются с целевыми значениями. В результате получают заключение о статистической управляемости или неуправляемости процесса [6].

Анализ жизненного цикла разработки ПО рассмотрен во многих источниках, однако нет детального описания управления каждым этапом жизненного цикла, включая тестирование [7].

Отдельное направление исследований — взаимосвязи между основными инструментами менеджмента качества и их влияние на операционные, финансовые и рыночные показатели компании-производителя. При этом интегрированное использование и управление таким инструментарием — актуальная задача [8].

В [9] показано, как различные практики управления качеством влияют на инновации в компаниях, сертифицированных по ISO 9001. Некоторые исследования посвящены учету практических рекомендаций при разработке жизненного цикла продукта. Из [10] известно, что 60–80 % потенциальных ошибок при разработке инновационных продуктов связаны с неправильно понятыми требованиями. Авторы [11] утверждают, что при обсуждении качества продукта следует прояснить вопросы качества кода, стоимости достижения заданного качества и времени выхода на розничный рынок. В вопросах автоматизации качества особая роль отводится непрерывной интеграции (или непрерывной доставке) программного кода [12]. Так называется процесс, который обеспечивает постоянное обновление кода при разработке софтверного продукта. В [13] даются рекомендации, как учитывать уровень конкуренции в отрасли при последовательном выпуске новых версий продукта.

Каждая из перечисленных работ рассматривает разные элементы управления качеством. При этом упускается из вида системное и структурное взаимодействие процессов. Как следствие, нет комплексного представления об управлении качеством при создании, тестировании и доработке ПО. Данное исследование призвано восполнить указанный пробел. Описывается комплексный подход, связывающий теорию, практику и методы управления качеством в ИТ-сфере. Отметим также, что благодаря гибкости предложенной модели ее можно адаптировать под нужды конкретного производителя.

Роль управления качеством при разработке ПО. Качество — это комплексная категория, которую можно рассматривать с разных точек зрения: философской, социальной, технической, экономической, правовой [1]. В данной статье качество обсуждается как совокупность свойств товара или услуги, способных в той или иной степени удовлетворять потребности целевой аудитории [3]. Продукт рассматривается как суммарный результат разных видов деятельности, причем у каждой — свои входные данные (параметры) которые в результате производственного процесса превращаются в выходные (ценность). Продукт — это совокупность ценностей. Он предназначен для удовлетворения потребностей, которые определяются заранее, при маркетинговой разработке товара или услуги. О качестве продукта судят по тому, насколько реальная удовлетворенность потребителя совпадает с запланированной.

В области информационных технологий основными считаются две группы требований к продукту (потребности пользователей): функциональные и нефункциональные. Как и на любом рынке, при розничной реализации ИТ-решений точно выстроенная работа с ценностью обеспечивает лояльность целевой аудитории, а значит, повышает рентабельность выпуска программного обеспечения, приложений и аналогичной продукции.

Реализованные дефектные товары портят имидж производителя [8]. Негативные отзывы распространяются в сети и мессенджерах. Потеря пользователей ведет к ухудшению финансовых показателей. Кризис качества может обернуться банкротством производителя.

В [6] показано, что менеджмент качества через управление процессами связан со всеми пятью типами инноваций. Перечислим факторы, которые их формируют:

- новая техника и технологии;
- продукция с новыми свойствами;
- новое сырье;
- новая организация производства;
- новые рынки сбыта.

Материалы и методы. Научные изыскания, результаты которых обобщены в представленной статье, базировались на глобальной практике производства и реализации софтверных товаров и услуг. Кроме того, изучалась литература, посвященная созданию и продвижению ИТ-решений. Теория соотнесена с практикой деятельности крупнейших разработчиков программного обеспечения. Теоретическая часть дает представление о специфике разных методов управления качеством. Приводятся примеры их практического использования при управлении жизненным циклом продукта. С этой точки зрения рассмотрены некоторые подходы к менеджменту проектов — гибкие, каскадные (второе название — водопадные) и гибридные.

Исследуется организация и координация релизов (выпусков) программного обеспечения. Это тесно связано с управлением качеством, однако редко или фрагментарно рассматривается в литературе.

Практика реализации крупных проектов соотносится с хорошо изученной теорией управления качеством. Это позволяет точно определить, как крупные организации задействуют теоретическую базу для обеспечения качества информационных продуктов (а в итоге — для их коммерческой успешности).

Отметим, что автор данной статьи имеет опыт управления софтверными продуктами, и его профессиональные знания тоже использовались в качестве материалов исследования.

Виды тестирования программного обеспечения. Как показано в [7], на рынке (то есть вне компании-производителя) восприятие качества (как элемента конкурентного преимущества) определяется в процессе проектирования продукта, статистического контроля и обратной связи. Представление о качестве внутри компании зависит от того, какой процент продукции проходит все проверки и в итоге не требует доработки. Этот показатель связан главным образом с управлением процессами, а не со статистическим контролем и обратной связью.

Важно, чтобы элементы системы, влияющие на качество, поддерживались высшим менеджментом и согласовывались с управлением персоналом. Кроме того, практики работы с качеством и его индикаторы нужно учитывать при настройках внутрикорпоративных отношений, взаимосвязей с поставщиками и другими контрагентами.

Для определения качества будущей и готовой продукции проводится тестирование, поэтому важно понимать, что именно является объектом испытаний. В данном случае недостаточно сказать: «продукт». Это слишком общее понятие, его следует конкретизировать. Часто к продукту предъявляют определенные функциональные и нефункциональные требования. В первом случае речь идет о наборе функций, которые система должна выполнять определенным образом. Например, при наборе адреса в строке браузера должна появиться соответствующая страница. Выбор определенных элементов меню на этой странице должен вести к заранее известным результатам. При этом может быть несколько сценариев взаимодействия с сайтом — например, для зарегистрированных и незарегистрированных пользователей. Так, большинство сетевых ресурсов при регистрации открывают возможность комментировать посты, получать подборки материалов и пр.

Как ясно из названия, нефункциональные требования не связаны с функциями, доступными пользователю. При этом они часто дают потребителю более весомые преимущества, чем функциональные. Речь может идти об определенном уровне защиты данных, комплексном сборе аналитики, соблюдении законодательства (например, о защите персональных данных).

Соответствие продукта функциональным и нефункциональным требованиям проверяется по-разному.

На рис. 1 представлена пирамида тестирования функциональных требований. Виды испытаний ранжированы по важности, скорости, стоимости и доле успешных проверок.

Рис. 1. Виды тестирования [14]

Юнит-тестирование, или модульное тестирование — это проверка корректности отдельных модулей исходного кода программы, отдельных функций приложения или сервиса. В результате устанавливается, работоспособен ли код, дает ли он правильные результаты при различных входных данных, соответствует ли логика ожидаемой. Пример такого теста: при сложении двух чисел должна получиться их сумма. В функцию вводятся входные значения, необходимые для проверки. Юнит-тест пройден, если ожидаемый результат функции равен фактически полученному. Для расчета метрики соотносят количество строк кода, прошедших тест, и общее число строк кода в репозитории. Лучшим индикатором считается 80 %. Он показывает, что логика любой функции соответствует ожидаемой на 80 % еще до того, как начинается следующая стадия тестирования. Значит, проблемы выявляются довольно рано и решаются гораздо дешевле и быстрее. Поэтому юнит-тест — первая ступень в пирамиде функционального тестирования. Его отсутствие (как и любых других видов тестирования) создает пробелы, накопление которых ведет к необоснованному удорожанию проекта [15].

Интеграционное тестирование. На следующем этапе тестируется взаимодействие компонентов будущего приложения. Проверяется интеграция модулей. Эти испытания сложнее юнит-тестирования, т. к. проверяется возможность выполнить последовательные действия в различных компонентах. Пример: пользователь входит в систему — перенаправляется на главную страницу — размещает объявление. То есть тестируются сразу несколько модулей. Ниже перечислены основные варианты интеграционного тестирования.

«Большой взрыв». Все компоненты собирают и совместно испытывают. Это позволяет выполнять тестирование один раз после разработки. Однако при большом количестве модулей какой-то можно упустить из виду, и это слабое место метода. К тому же увеличивается петля обратной связи, так как тестируются готовые решения, когда разработка завершена. Отметим также сложности с локализацией ошибки. Они объясняются тем, что придется проанализировать весь процесс (сценарий) разработки, чтобы узнать причину проблемы. Это значит, что время тестирования увеличится. Метод однозначно удобен для небольших систем.

Инкрементальный подход. В отличие от «большого взрыва» испытания можно начинать, даже если к нему готовы всего два модуля, и добавлять остальные по мере их разработки. В этом случае проще локализовать исходные ошибки, сокращается петля обратной связи. Не нужно ждать, пока будут готовы все сервисы. А значит, проверка начнется и завершится раньше. Устранение дефектов обойдется дешевле. Инкрементальное тестирование можно выполнять двумя способами: снизу вверх и сверху вниз.

Интеграция снизу вверх. Сначала тестируются самые некритичные модули, затем более значимые и наконец — базовые. Следовательно, верхние компоненты нельзя проверить, пока нет данных о качестве элементов нижнего уровня. Это увеличивает время между разработкой и стартом тестирования.

Интеграция сверху вниз. Тестирование начинается с наиболее значимых уровней приложения. В данном случае испытание модулей нижних уровней может быть не вполне адекватным. Однако даже если впоследствии это обернется проблемами, они будут некритичными.

Гибридная интеграция. Модули разных уровней тестируются совместно. Успешность такой проверки определяется тем, понимает ли исполнитель архитектуру приложения и может ли определить оптимальный подход к конкретному продукту [12].

Системное тестирование. Все компоненты программы испытываются как единое приложение. Специалист должен удостовериться, что продукт корректно отрабатывает различные сценарии и ситуации. Этот вариант может включать тестирование нефункциональных требований. Такой подход основывается на требованиях или на базе сценариев использования. В любом случае результатом будут тестовые сценарии. Их успешное прохождение подтверждает, что система работает именно так, как ожидается.

Системное тестирование требует значительного времени, поскольку с развитием продукта растет количество возможных вариантов ее использования. Следовательно, софт нужно своевременно обновлять. Этим занимается служба поддержки. При каждом обновлении продукта необходимо проверить, работают ли ранее протестированные функции. Этот процесс называется регрессионным тестированием. Если ранее тесты работали, а после ввода новой функции перестали, значит, есть проблема. Процедура такой проверки иногда занимает колоссальное количество рабочего времени инженеров по тестированию, и они не успевают писать новые тесты. Этим обусловлена важность автоматизации испытаний, которые выполняются вручную. При таком подходе тестирование оптимизируется до простого анализа результатов автоматических проверок. Лучший вариант — это 100-процентная автоматизация. Достичь такого показателя не всегда возможно, поскольку некоторые тесты не подлежат автоматизации из-за их сложности [16].

Приемочное тестирование. На последнем этапе продукт тестируют ключевые специалисты и заказчики. Они в режиме реального времени проверяют, насколько реализованы их ожидания и требования. Выясняется, с какими проблемами столкнется будущий пользователь. Все ли функции работают, как было задумано, удобен ли интерфейс, насколько вероятны ошибки и некорректные операции. В итоге проект принимается или не принимается. Приемка — это заключительный этап тестирования.

Исправление недоработок и дефектов, обнаруженных на данном этапе, обойдется особенно дорого, поскольку для решения проблемы придется пройти весь процесс с самого начала — от разработки и юнит-тестирования до полной проверки системы. Следовательно, на более ранних стадиях нужно приложить все усилия, чтобы избежать такой ситуации.

Нефункциональное тестирование проверяет приложение на:

- производительность;
- надежность;
- совместимость;
- безопасность;
- полезность;
- масштабируемость.

Для проверки этих свойств используют более 15 видов тестирования. Назовем наиболее распространенные.

Тестирование производительности. Оценивается скорость и эффективность работы приложения или системы. Основное внимание уделяется скорости ответа на запрос пользователя. Например, тестирование может включать оценку времени загрузки основной страницы сайта. Определяются целевые показатели производительности, и с ними сравниваются реальные результаты.

Нагрузочное тестирование. Выясняется максимальная нагрузка, которую выдерживает система без отказа или существенного ухудшения производительности. Например, для сайта это означает определение количества пользователей, которые могут одновременно взаимодействовать с платформой, прежде чем произойдут сбои или отказы.

Тестирование отказоустойчивости. Испытывается способность системы или приложения сохранять работоспособность при сбоях или непредвиденных ситуациях. Цель такого тестирования — обнаружение уязвимостей и разработка мероприятий для обеспечения непрерывной работы.

Тестирование совместимости. Проверяется совместимость приложения или системы с другим программным обеспечением, с различными операционными системами, браузерами и устройствами.

Тестирование безопасности. Оценивается уровень защищенности системы или приложения от атак и потенциальных угроз безопасности. Выясняется, есть ли уязвимости, анализируются защитные механизмы и эффективность мер безопасности.

Тестирование аварийного восстановления. Определяется способность системы или приложения восстанавливаться после сбоев, отказов или других чрезвычайных ситуаций. Цель — проверка эффективности процедур восстановления и минимизация времени простоя в случае аварийных ситуаций.

Результаты исследования. В рамках представленного исследования сформирована комплексная модель управления качеством при создании ПО. Описаны ее взаимосвязи с жизненным циклом разработки, релизами, издержками и моделью менеджмента проектов (гибкой, каскадной или гибридной).

Процесс управления качеством в жизненном цикле разработки ПО. Стандартная разработка цифровых продуктов состоит из шести производственных процессов, или этапов. Каждый из них должен вносить вклад в обеспечение качества (рис. 2).

Рис. 2. Жизненный цикл разработки программного обеспечения [17]

Анализ и определение требований к продукту. Исследуются проблемы, предлагаются решения с учетом заявленных требований. Этот процесс задает основные показатели для проверки готового продукта. В зависимости от модели разработки описываются и согласовываются стратегия, подходы и инструменты тестирования. Один из документов называется «Видение продукта и его возможности». В нем фиксируются выявленные проблемы, отмечаются связи с бизнес-планом, описывается конъюнктура рынка, перечисляются основные функциональные и нефункциональные требования. Документ утверждают стороны-партнеры (то есть представители исполнителя и заказчика). При необходимости в текст вносят корректировки. На этом этапе важно привлечь специалистов по безопасности и архитектуре приложений, которые помогут точно определить нефункциональные требования, важные для будущего тестирования.

Дизайн архитектурного и визуального решения. С учетом требований, выявленных на предыдущей стадии, вырабатывается архитектурное решение, обеспечивающее их выполнение, и дизайн пользовательского интерфейса. Все это — ориентир для команды разработчиков, которым предстоит тестировать и при необходимости корректировать продукт.

Разработка. Создается код для решения задач и проблем будущих пользователей. Для проверки логики отдельных функций пишутся юнит-тесты. Основная цель — достичь нужного показателя покрытия кода тестами (не менее 80 %). Однако широко применяются и другие практики — такие как аудиты кода (code review). В этом случае код проверяют другие члены команды, и на ранних стадиях разработки выявляются ошибки, недочеты, пропуски и уязвимости. Ревьюеры могут оставлять комментарии к конкретным строкам кода, указывая на необходимость переработки. С точки зрения качества продукта идеальный сценарий — двойное код-ревью, когда минимум два не связанных с автором специалиста рассматривают и комментируют код.

Кроме юнит-тестирования и код-ревью необходимо использовать линтеры. Это автоматизированные инструменты, которые анализируют код в реальном времени, обнаруживают недочеты, ошибки и предлагают пути их исправления. Можно использовать и другие виды тестирования в зависимости от стратегии и требований к продукту.

Тестирование. Инженеры по тестированию разрабатывают адекватные задачам тест-сценарии. Как правило, они пишутся вручную, что замедляет реализацию данного этапа. Целесообразно автоматизировать процесс, однако это не всегда возможно. На данной стадии специалисты проверяют и при необходимости перепроверяют результаты ручных и автоматизированных тестов. Особое внимание уделяется случаям, когда автоматические тесты выявляют ошибки. Важно установить, реальная ли это ошибка, или ложноположительный результат. Обнаруженные дефекты ранжируются в зависимости от их важности для системы или пользователей, а затем передаются разработчикам для исправления. После устранения дефектов полезно провести анализ корневой причины проблемы с использованием метода RCA (англ. root cause analysis — анализ причин). Это позволяет разработать меры по предотвращению проблем в будущем, добавить новые юнит- или интеграционные тесты. Для полного понимания причин проблемы рекомендуется использовать технику «5 почему» (англ. 5 Why’s). Схема метода выглядит так: формулируется проблема и задается вопрос: «Почему это случилось?». На него следует ответ, описывающий определенный факт или ситуацию. Вновь задается вопрос: «А это почему случилось?». И так пять раз. Пять ответов позволяют приблизиться к пониманию причин проблемы.

Тестирование — это масштабный процесс. Ниже перечислены его основные этапы.

1. Настройка и конфигурация окружения для тестирования.
2. Написание и выполнение ручных тест-сценариев.
3. Автоматизация ранее написанных ручных тест-сценариев.
4. Проверка результатов автоматических тестов.
5. Поддержка актуальности автоматических тест-сценариев и исправление ошибок.
6. Фиксация дефектов.
7. Документирование дефектов.
8. Анализ причин дефектов.
9. Разработка шагов по предотвращению появления дефектов в будущем.

Развертывание кода приложения. Программное обеспечение требует различных сред для развертывания. Важно хорошо представлять, как непрерывная интеграция и непрерывное развертывание (англ. continuous integration and continuous deployment, CI/CD) обеспечивают стабильное качество ПО при минимальных трудозатратах. Этому способствует в первую очередь автоматизация развертывания кода — перенос его изменений из одной среды в другую. Процесс делится на этапы, которые зависят от продукта, организации и принятых стандартов.

Степень автоматизации, количество и строгость проверок определяют ожидаемое качество продукта до того, как код станет доступен конечным пользователям. На рис. 3 показана возможная схема развертывания кода. При высоком уровне автоматизации трудозатраты будут минимальными, придется выполнить вручную всего несколько проверок.

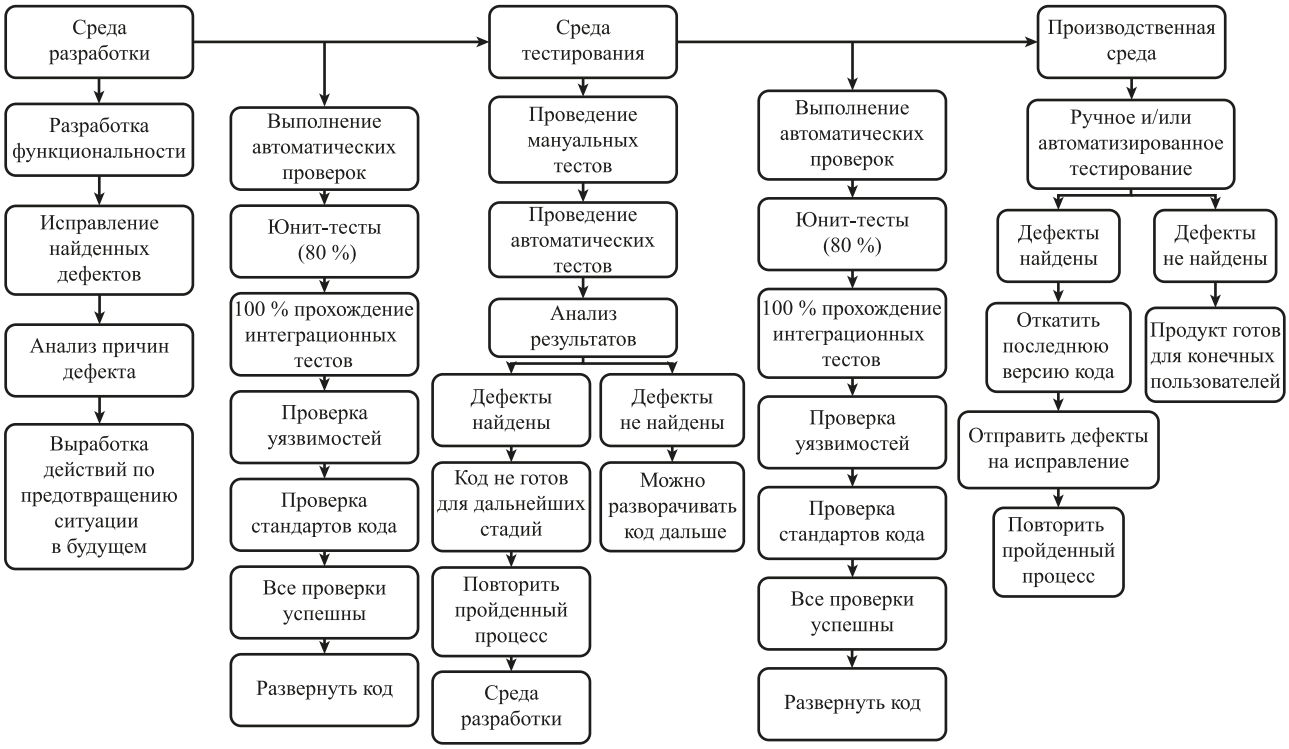

Рис. 3. Схема процессов и проверок при развертывании кода

Следуя представленной схеме, можно контролировать качество и автоматизировать развертывание кода. Это позволит сократить время на разработку, а значит, продукт быстрее выйдет на рынок.

Поддержка продукта. На этом этапе необходимо выстроить обратную связь с пользователями: проводить опросы, собирать и анализировать метрики. Все это будет базой для задач и идей по улучшению ПО. Предположения следует подтвердить A/B тестами, т. е. сравнением вариантов продукта. Важно также обеспечить обратную связь с командой поддержки, которая напрямую контактирует с пользователями. Именно эти специалисты делятся ценными наблюдениями, передают пожелания целевой аудитории, на основании которых проводится анализ и формируются новые стратегии. Как видим, процесс не завершается, даже если продукт полностью готов. Появляются новые данные для анализа, сбора требований и развития цифрового товара или услуги.

Обсуждение и заключение. Представленная в статье модель управления качеством адаптируется под нужды различных организаций и продуктов. Можно дополнительно изучить ее компоненты, чтобы создать оптимальный план для достижения целей, связанных с управлением качеством. При этом следует иметь в виду ключевые факторы коммерческого успеха ПО: разработчики и тестировщики должны критически мыслить, наладить сотрудничество и постоянно совершенствовать процессы.

Предложенная модель предполагает стратегические изменения, реализация которых требует значительного времени, поэтому общий процесс трансформаций следует разбить на части.

Изложенный материал будет полезен менеджменту, который курирует вопросы качества, касающиеся общей стратегии компании и оптимизации отдельных процессов.

Отметим, что для достижения значимых результатов необходимы высокая степень автоматизации тестирования, развитая инженерная культура и значительные затраты на создание и поддержку инфраструктуры.

Список литературы / References

1. Suma V, Gopalakrishnan Nair TR. Defect Management Strategies in Software Development. In book: *Recent Advances in Technologies*. Vienna: Intec Web Publishers; 2009. P. 379–404. https://doi.org/10.48550/arXiv.1209.5573

2. Кузнецов Л.А. Управление качеством в сложных технологических процессах. *Проблемы управления.* 2007;(3):47–53.

Kuznetsov LA. Quality Management in Complex Processes. *Control Sciences.* 2007;(3):47–53.

3. Munoz M, Mejia J, Ibarra S. Tools and Practices to Software Quality Assurance: A Systematic Literature Review. In: *Proc. 13th Liberian Conference on Information Systems and Technologies* (*CISTI*). New York City: IEEE; 2018. P. 1–6. https://doi.org/10.23919/CISTI.2018.8399334

4. Carrozza G, Pietrantuono R, Russo S. A Software Quality Framework for Large-Scale Mission-Critical Systems Engineering. *Information and Software Technology.* 2018;102(3):100–116. https://doi.org/10.1016/j.infsof.2018.05.009

5. Vallabhaneni RS. Corporate Management, Governance, and Ethics Best Practices. Ch. 7. In book: *Quality-Management Best Practices*. Hoboken, NJ: Wiley; 2008. 456 p. https://doi.org/10.1002/9781119196662.ch7

6. Размочаева Н.В., Семенов В.П., Безруков А.А. Исследование статистического управления процессами в задачах автоматизации процессов. В: *Материалы XII Международной конференции по мягким вычислениям и измерениям*. 2019;1:355–358. URL: https://scm.etu.ru/assets/files/2019/scm2019/papers/7/355.pdf (дата обращения: 20.06.2024).

Razmochayeva NV, Semenov VP, Bezrukov AA. Investigation of Statistical Process Control in Process Automation Tasks. In: *Proc. XII International Conference on Soft Computing and Measurements.* 2019;1:355–358. (In Russ.) URL: https://scm.etu.ru/assets/files/2019/scm2019/papers/7/355.pdf (accessed: 20.06.2024).

7. Невлюдов И.Ш., Андрусевич А.А., Евсеев В.В. Анализ жизненного цикла разработки программного обеспечения для корпоративных информационных систем. *Восточно-Европейский журнал технологий предприятий*. 2010;6(8):25–27.

Nevlyudov ISh, Andrushevich AA, Evseyev VV. Software Development Life Cycle Analysis for Enterprise Information Systems. *Eastern European Journal of Enterprise Technologies.* 2010;6(8):25–27. (In Russ.)

8. Kaynak H. The Relationship between Total Quality Management Practices and Their Effects on Firm Performance. *Journal of Operations Management.* 2003;21(4):405–435. https://doi.org/10.1016/S0272-6963(03)00004-4

9. Dong-Young Kim, Vinod Kumar, Uma Kumar. Relationship between Quality Management Practices and Innovation. *Journal of Operations Management*. 2012;30(4):295–315. https://doi.org/10.1016/j.jom.2012.02.003

10. Ramasubbu N, Kemerer CF. Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests. *IEEE Transactions of Software Engineering*. 2019;45(3):285–300. https://doi.org/10.1109/TSE.2017.2774832

11. Alhassan A, Alzahrani W, AbdulAziz A. Total Quality Management for Software Development. *International Journal of Computer Applications.* 2017;158(5):38–44. URL: https://www.ijcaonline.org/archives/volume158/number5/alhassan-2017-ijca-912850.pdf (accessed: 20.06.2024).

12. Mohamed SI. Software Release Management Evolution — Comparative Analysis across Agile and DevOps Continuous Delivery. *International Journal of Advanced Engineering Research and Science*. 2016;3(6):52–59. URL: https://ijaers.com/detail/software-release-management-evolution-comparative-analysis-across-agile-and-devops-continuous-delivery/ (accessed: 20.06.2024).

13. Adelman D, Mancini A. Dynamic Release Management: A Market Intensity Approach. *Chicago Booth Research Paper*. 2016;(16–19):42. http://doi.org/10.2139/ssrn.2847264

14. Radziwill N, Freeman G. Reframing the Test Pyramid for Digitally Transformed Organizations. *Semantic Scholar*. URL: https://www.semanticscholar.org/reader/62a5c71b33437bc40e146a13a6fb95371b866262 (accessed: 22.06.2024).

15. Alves NSR, Mendes TS, Mendonca MG, Spinola RO, Shull F, Seaman C. Identification and Management of Technical Debt: A Systematic Mapping Study. *Information and Software Technology.* 2016;70(2):100–121. https://doi.org/10.1016/j.infsof.2015.10.008

16. Concas G, Marchesi M, Murgia A, Tonelli R, Turnu I. On the Distribution of Bugs in the Eclipse System. *IEEE Transactions on Software Engineering*. 2011;37(6):872–877. http://doi.org/10.1109/TSE.2011.54

17. Lemke G. The Software Development Life Cycle and Its Application. *Senior Honors Theses and Projects*. 589. Ypsilanti, MI: Eastern Michigan University; 2018. URL: https://commons.emich.edu/cgi/viewcontent.cgi?article=1588&context=honors (accessed: 22.06.2024).

Об авторе:

Мартин Дмитриевич Бируля, менеджер проектов EPAM Systems Sp z o.o. (Польша, 31–553, г. Краков, ул. Опольска, 114), SPIN-код, ORCID, martinbirulia@gmail.com

Конфликт интересов: **автор заявляет об отсутствии конфликта интересов.**

Автор прочитал и одобрил окончательный вариант рукописи.

About the Author:

Martin D. Birulia, project manager, EPAM Systems Sp z o.o. (114, Opolska Str., Krakow, 31–553, Poland)**,** SPIN-code, ORCID, martinbirulia@gmail.com

Conflict of Interest Statement: **the author claimed no conflict of interest.**

The author has read and approved the final manuscript.

Поступила в редакцию / **Received** 30.06.2024
Поступила после рецензирования / **Reviewed** 20.07.2024
Принята к публикации / **Accepted** 27.07.2024