УДК: 004

1.9. Методология управления кодовой базой программных комплексов в условиях цифровой экономики

Зекирьяев Р.Т., Болбаков Р.Г.
МИРЭА – Российский технологический университет, Москва 119454, Россия

Управление IT-проектами, наряду с управлением проектами в отраслях промышленности с более долгой историей, таких как металлургия, машиностроение, тяжёлая промышленность, является регламентированным процессом, в котором все этапы регулируются различными документами – методологиями, соглашениями, стандартами. Эти регламенты помогают использовать всем участникам процесса практики, зарекомендовавшие свою эффективность – начиная с этапов планирования проекта, завершая тестированием и выводом проекта в эксплуатацию. По мере становления различных процессов, появляются новые методологии, вбирающие в себя весь накопленный опыт и дающие возможность этим опытом воспользоваться. Жизненный цикл IT-проекта состоит из множества этапов, каждый из которых описан либо отдельно в соответствующем стандарте, либо в составе комплексного описания группы этапов. Этап разработки программного обеспечения представляет собой временной промежуток, за время выполнения которого создаётся основная ценность создаваемого решения, поэтому процессы работы с исходным кодом являются одними из наиболее важных среди всех процессов жизненного цикла. На текущий момент произошло становление очередного ряда процессов этапа разработки, описание и формализация которых дает возможность большому числу разработчиков пользоваться лучшими практиками и не изобретать велосипед в работе с кодовой базой. В научной статье рассматривается история становления методологий в сфере информационных технологий и пути формализации используемых практик. Затем, даётся краткая характеристика состояния сферы в контексте работы с исходным кодом программного продукта. Следующим шагом описываются зарекомендовавшие себя практики работы с кодовой базой, на их основе выполнена систематизация накопленных знаний и приведение к виду методологии, готовой к практическому применению. Пример использования описанной в работе методологии служит для подтверждения её жизнеспособности и обозначения границ её практического применения.

Введение

В конце 60-х годов прошлого века в разработке программного обеспечения стали проявляться характерные черты надвигающейся проблемы. На фоне быстрого развития средств вычислительной техники росла сложность создания и поддержки программного обеспечения для этой техники. В 1968 г. НАТО была проведена первая конференция, посвящённая проблемам разработки программного обеспечения – Software Engineering [1]. Через год, в 1969 г. состоялась вторая конференция - Software Engineering Techniques [2]. На этих конференциях сложившуюся ситуацию назвали кризисом программного обеспечения или в оригинале – «‘software crisis’ or the ‘software gap’». И именно тогда ведущие мировые эксперты в области разработки программного обеспечения пришли к выводу, что нестрогие методы разработки программного обеспечения требуют формализации. Возможно, именно эти конференции дали толчок к появлению методологий разработки программного обеспечения в том виде, в котором мы их знаем теперь, они описаны в стандартах программной инженерии [3, 4], в Scrum-guide [5] и в других документах [6].

Этот небольшой исторический экскурс даёт понять, что из кризисной ситуации, сложившейся в индустрии разработки программного обеспечения был найден выход – путём формализации уже существующих подходов. Разумеется, такое решение было принято и для того, чтобы предотвратить возникновение подобных ситуаций в дальнейшем. Так начался путь становления методологий работы с программным обеспечением.

Суть методологий разработки заключается в том, чтобы объединить лучшие практики, существующие на данный момент и описать способы их применения. Существующая на текущий момент времени методология может быть неприменима через десять лет, однако это не значит, что нужно отказываться от неё сегодня.

Наглядным примером развития методологий управления является переход, который мы видим в наши дни – от каскадной модели управления проектами (Waterfall) [6] к гибким методологиям семейства Agile [7].

Такое развитие даёт понять, что если сегодня мы имеем набор практик и видим его применение при масштабировании, то однозначно следует пытаться описывать эти практики и способы их применения.

Перед тем, как перейти к рассмотрению основной части работы, сформулируем гипотезу – на текущий момент отсутствуют единые стандарты, охватывающие полный цикл работы с кодовой базой в повседневной профессиональной деятельности.

Далее постараемся подтвердить или опровергнуть эту гипотезу.

Методология работы с кодовой базой

Первым делом рассмотрим используемые в работе термины, способные трактоваться разными способами. В рамках статьи под «кодовой базой» понимается совокупность файлов с исходным кодом, написанным человеком. Файлы, сгенерированные средой разработки, средствами сборки и другими инструментами на основе написанного разработчиком кода не относятся к кодовой базе.

Итак, на текущий момент работа над IT-проектом разбивается на следующие этапы – анализ и описание требований, создание дизайна, разработка и тестирование кода, внедрение проекта в эксплуатацию. По существу, эти этапы – практическое упрощение технических процессов жизненного цикла системы из «ГОСТ Р 57193-2016 Системная и программная инженерия. Процессы жизненного цикла систем» [3] или его аналога «ISO/IEC 15288:2015 Systems and software engineering — System life cycle processes» [4]. Это серьёзные и очень подробные стандарты, однако большинство участников IT-проектов не знакомо с их полным содержанием, так как на практике используются только описанные выше этапы, зачастую декомпозированные или обобщённые.

Для каждого из описанных этапов существуют свои наборы практик и методологий. Некоторые выступают контейнерами – такие как Scrum [8], другие описывают конкретный этап – такие как пирамида тестирования [9].

На сегодняшний день отсутствуют общие практики по работе с кодовой базой, несмотря на наличие таковых на локальных уровнях. Практики повторяются от команды к команде, но если команда сформировалась в небольшом временном отрезке (к примеру - вчера), то все эти практики, с высокой вероятностью, команде придётся создавать с нуля. Такие практики необходимы на каждом проекте, где над кодовой базой работает более одного человека. И даже если над кодом работает один человек, то общие операции с кодовой базой и практики работы с ней всё равно должны быть на случай, если единственный разработчик сменится. Всё это необходимо также для того, чтобы придерживаться подхода формализации, который было решено применить к сфере разработки на конференции по программной инженерии более пятидесяти лет назад. Если не описывать и не обобщать практики и операции по работе с кодом, то мы рискуем оказаться в новом кризисе программного обеспечения, который будет вызван не разрывом по прогрессу с вычислительной техникой, а простым запутыванием систем и наплывом информационного шума [10].

Программное обеспечение, созданное несколько лет назад и находящееся в постоянном развитии, представляет собой гигантские объёмы кода, которые, без принятия должных мер по описанию практик работы с ними, будут требовать всё больше ресурсов на поддержание и развитие, а вновь привлечённые ресурсы будут вызывать ещё большее запутывание. Без описания хотя бы каким-нибудь способом практик работы с кодовой базой, эта база при смене разработчиков, создающих её, рискует очень быстро перейти в категорию legacy [11, 12] (устаревшая, тяжёлая в поддержке система).

Существующие практики работы с кодовой базой

Для решения описанных проблем создаются практики по поддержанию и развитию кодовой базы программных продуктов. Рассмотрим эти практики подробнее.

Разработка программного продукта зачастую выполняется совместно группой разработчиков. Чтобы избежать проблем с внесением параллельных доработок в общее хранилище с исходным кодом и описать процесс внесения этих доработок в кодовую базу используют модели ветвления. Наибольшей популярностью, пользуются такие модели, как Feature Based Development (FBD), Trunk Based Development (TBD) [13, 14]. Идеологии этих моделей зачастую противопоставляются друг другу, в отличие от предметной области, охватываемой ими. Модель ветвления, или Branching Model, описывает, как работать с системой контроля версий при выполнении локальных доработок, и как затем эти доработки сливать с кодовой базой. Модель ветвления описывает начало и завершения процесса создания программного инкремента.

Сам процесс работы и соглашения, принятые в проекте описываются с использованием другой практики – Code style (CS) [15]. В CS описывается стандарт оформления кода, взаимное расположение различных семантический конструкций, а также любые другие особенности работы с исходным кодом.

Кроме локальных CS для проекта существуют общепринятые соглашения на уровне языка программирования. Примерами таких CS являются Coding conventions для Kotlin [16] или PEP 8 – Style Guide for Python Code для Python [17]. Соглашения на уровне языка программирования имеют большую популярность и зачастую используются на проектах без изменений, что положительно сказывается на общем состоянии кода и обеспечивает повсеместность стилей его написания. CS описывает непосредственно процесс работы с кодом.

Ещё одной важной практикой при работе над одним проектом группой разработчиков является Code Review (CR) – проверка вносимых изменений другими разработчиками проекта [18]. Если открыть самые популярные репозитории за месяц на GitHub, то можно обнаружить, что примерно в 9 из 10 случаев для репозитория характерно использование CR в процессе Pull Request'ов и слияния доработок с исходным кодом [19]. CR описывает последний этап работы над инкрементом – его слияние с основной кодовой базой.

Разумеется, существует ещё множество более или менее популярных практик для работы с кодовой базой программных продуктов, однако их объединяет одно особенность – они описывают только определённый этап создания инкремента. Модель ветвления описывает отведение и слияние локальных доработок, CS содержит информацию по оформлению кода, а CR проводится на этапе слияния доработок. Разработка — это не какой-либо отдельный процесс из описанных выше, но совокупность всех этих практик.

Таким образом, мы подтверждаем гипотезу о том, что на текущий момент отсутствует единый стандарт, описывающий полный цикл работы с кодовой базой.

Опыт развития индустрии разработки программного обеспечения, описанный в начале работы, показывает, что возникающие проблемы могут привести к серьёзным последствиям, при отсутствии методологий, способных формализовать все этапы процесса работы с кодовой базой. Поэтому, рассмотрим совокупность рассмотренных разрозненных практик, как методологию по управлению кодовой базой.

Методология Codebase Operations and DEvelopment practices (CODE)

Работая над разными проектами, наблюдая разные подходы, авторы настоящей научной статьи пришли к идее создания методологии управления кодовой базой CODE – Codebase Operations and DEvelopment practices.

Эта методология описывает процесс дополнения общей кодовой базы, она позволяет увеличить продуктивность работы с кодовой базой проекта, даёт возможность писать поддерживаемый и масштабируемый код, а также снижает риски нарушения процессов, построенных вокруг работы с кодом.

В чём заключается суть методологии? Возьмём условного разработчика программного продукта, работающего в команде. Работая над своей частью продукта, он выполняет одну и ту же последовательность действий, как для разработки новой функциональности, так и для поддержки или внесения правок в текущую. Значит, поскольку это цикличный процесс, то его можно декомпозировать на повторяющиеся этапы. Декомпозировав этапы до удобных масштабов, мы можем описать их. Описанные этапы, в свою очередь, представляют собой в совокупности отличную инструкцию.

Так как речь идёт непосредственно о работе с кодом, то выделим четыре больших этапа, каждый из которых рассмотрим подробнее: Prepare (Подготовка), Develop (Разработка), Control (Проверка), Apply (Применение изменений).

Цикличный процесс по работе с продуктом нужен для поддержания качества разработки, что наталкивает на аналогию с принципом Деминга-Шухарта [20, 21], который заключается в последовательности Plan-Do-Check-Act. Представлен на рисунке 1.

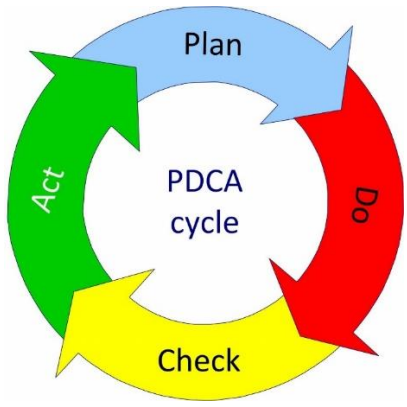

Рисунок 1 – Цикл PDCA Деминга-Шухарта

Цикл работы с кодом в рамках CODE имеет аналогичную структуру и родился, как имплементация цикла управления качеством.

Так, описание операций с кодовой базой и практик разработки сводится к описанию каждого из четырёх этапов цикла PDCA. В рамках методологии CODE цикл PDCA имеет описанную ранее структуру: Prepare-Develop-Control-Apply. На первом этапе выполняется подготовка к разработке, подготовка включает в себя информацию, необходимую уже непосредственно при работе с кодом. На втором этапе методологии, выполняется разработка по определённым правилам, которые также нужно иметь ввиду. Третий этап – проверка, он включает в себя принятые практики анализа написанного кода. Четвёртый этап включает в себя применение созданных изменений. Сам цикл представлен на рисунке 2. Рассмотрим, что

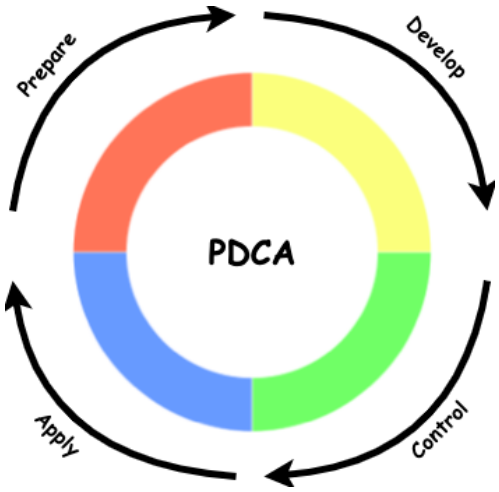

Рисунок 2 – Цикл Prepare-Develop-Control-Apply

конкретно включает в себя каждый из этапов.

1. Prepare (Подготовка)

При работе с большими объёмами кода используются системы контроля версий. Для эффективной работы с системой в параллели с одной кодовой базой используются различные модели ветвления (Branching) – Feature Based Development (FBD), Trunk Based Development (TBD). Очень важно, чтобы все разработчики понимали, какая модель используется в их кодовой базе, как выполнять доработки и слияние своих доработок с общей кодовой базой, сколько коммитов должно быть в локальной ветке. Поэтому самым первым шагом является изучение модели ветвления и всех принятых практик по работе с системой контроля версий – отведение релизных веток, хотфиксов [22] и простых доработок. Основа этого этапа – описанные практики работы с системой контроля версий.

2. Develop (Разработка)

После выполнения подготовки, создания собственного пространства для работы в системе контроля версий, наступает черёд непосредственно разработки. Разработка также не должна носить хаотичный характер – поэтому при написании кода хорошей практикой является следование стилю написания кода или Code style (CS). Code style содержит в себе информацию о именовании сущностей в проекте, расположении свойств и методов в рамках класса, рекомендации по оформлению кода и написанию комментариев. В рамках Code style также могут быть даны рекомендации по написанию методов или реализации задач, специфичных для конкретной кодовой базы. Здесь же могут описываться используемые в разработке шаблоны проектирования. Ключевой регламент этапа разработки – стиль написания кода.

3. Control (Проверка)

За разработкой следует этап проверки или тестирования. На этом этапе код уже приобрёл бизнес-ценность [23] и её следует проверить на соответствие ожиданиям. В первую очередь выполняется ручная проверка разработанной функциональности. Далее на этом же этапе выполняется автоматизации тестирования через написание тестов. Например, хорошей практикой будет следование упомянутой ранее пирамиде тестирования – написание малой доли End to end (E2E) тестов, чуть большего числа интеграционных тестов и большого числа, относительно предыдущих двух типов, Unit-тестов. На этом этапе следует описать практики автоматизированного тестирования.

4. Apply (Применение изменений)

Завершающий этап цикла – применение разработанных изменений. В соответствии с практиками работы с системой контроля версий создаётся запрос на слияние с основной кодовой базой созданных изменений (Pull/Merge Request). Создание таких запросов также желательно описывать, чтобы избежать различий в форматах. Название, описание, пояснения к доработкам, размер изменений в одном запросе. За этапом создания запроса следует Code Review (CR) – проверка вносимых изменений другими разработчиками проекта. CR может быть очень сложным процессом, поэтому следует заранее договориться о правилах его проведения – просматривать поступившие запросы на CR за конкретный, ограниченный во времени срок, абстрагироваться от эмоционального контекста, если он мешает качественному просмотру доработок.

Применение методологии CODE в индустриальном производстве

Рассмотрим пример реализации CODE для проекта, который сымитирует реальный индустриальный проект. Пусть команда разработчиков из 10 человек работает над продуктом на несколько сотен тысяч строк кода. На проект берут нового разработчика. Что ему нужно знать, когда он начнёт работать с кодовой базой, какие операции он будет выполнять ежедневно, каким набором практик ему следует руководствоваться? Методология CODE даст ответы на указанные вопросы.

Далее обратим внимание на цикл улучшения качества CODE для этого проекта.

Prepare. Используется git с моделью ветвления TBD. Основная ветка – master. Для доработок создаётся ветка от master формата feature/XXXX-yyyy, где XXXX – номер реализуемой задачи в трекере, а yyyy – краткое описание на английском языке. Ветка сливается с основной в течение 1-7 суток, одна ветка содержит один коммит.

Develop. Code style согласно установленному разработчиками языка в среде разработки. Линтер включает проверки на соответствие Code style. Все новые методы описываются многострочным комментарием перед телом метода. Доработки декомпозируются и закрываются локальными feature-флагами, если функциональность ещё не готова.

Control. Проверку доработанной функциональности выполняет разработчик, затем она передаётся тестировщикам. Процент покрытия кода Unit-тестами не менее 80%. Если разрабатываемая функциональность затрагивает более одного модуля приложения, она включается в интеграционные и E2E тесты.

Apply. Слияние изменений через Merge Request. Название содержит номер задачи и описание. Если изменения затрагивают более 20 файлов, добавляется комментарий с описанием изменений. Merge Request проходит Code Review (CR), минимальное число ревьюеров/проверяющих – 2. CR выполняется не более, чем сутки. В ходе CR не рекомендуется выказывать личную неприязнь и оскорбления – иначе проверяющий заменяется.

Это краткий вариант реализации рассмотренной методологии CODE. Ознакомившись с ней, разработчик начинает понимать, с чем ему предстоит работать.

На рисунке 3 показан вариант хранения файла с описанием методологии на примере Android-проекта.

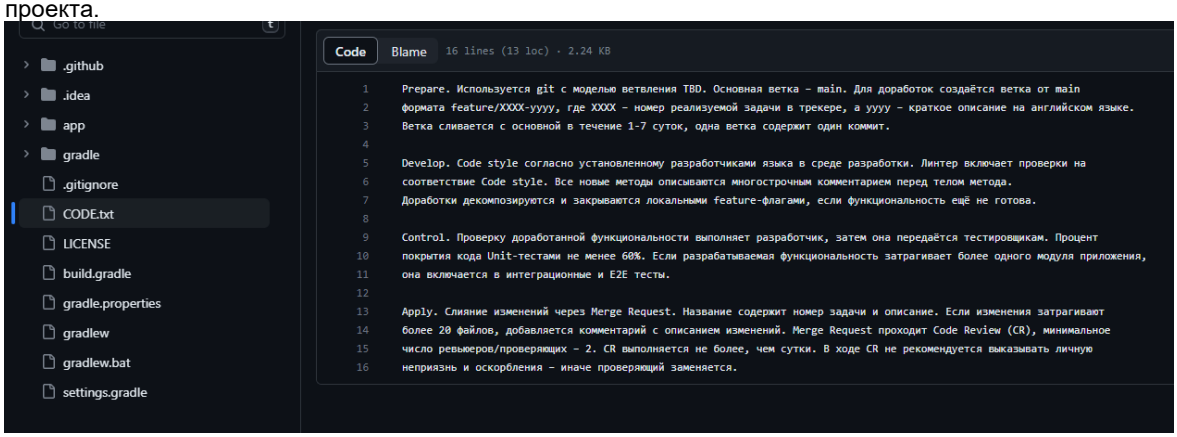

Рисунок 3 – CODE.txt в репозитории проекта

Крайне важной частью принятия методологии является ознакомление с ней всех разработчиков, поддерживающих конкретную кодовую базу – для этого очень важно формулировать пункты простым языком, без неопределённых терминов или, если таковые неизбежны, с поясняющими ссылками. Также упрощает взаимодействие с документом чёткое перечисление пунктов и их краткое изложение. Наличие практических примеров тоже очень рекомендуется, так как оно даёт понимание, куда применять все описанные материалы. Никто не любит читать огромные массивы текста, написанные сложным языком с неясной читателю целью.

Именно эти пункты представляют собой непрекращающийся цикл разработки, поэтому их знание очень важно для каждого разработчика, взаимодействующего с кодовой базой. С этими четырьмя пунктами разработчик сталкивается каждый день, поэтому CODE находится в постоянном совершенствовании и дополнении.

Перспективы применения методологии CODE

Можно говорить о положительном эффекте применения методологии для самых разных сфер, в которых задействованы информационные технологии, однако наиболее точно понять суть внедрения поможет рассмотрение на примере крупных и всеобъемлющих сфер. Одной из таких сфер является сфера повсеместной цифровизации, утверждённая от 2019 года в национальной программе «Цифровая экономика Российской Федерации» [24]. Развитие цифровых технологий и обеспечение нормативного регулирования цифровой среды являются ключевыми направлениями развития цифровой экономики, что прямо свидетельствует об актуальности рассматриваемой темы.

Крупнейшие технологические компании, представленные на отечественном рынке, обладают огромными ресурсами, связанными с проектами в сфере информационных технологий. Так, в Сбере работает около 45 тысяч ИТ-специалистов [25], а размер кодовой базы Яндекса превышает 45 ГБ [26]. Даже поверхностная количественная оценка ресурсов компаний свидетельствует о необходимости применения методов нормативного регулирования ИТ-процессов. Кроме того, сквозная стандартизация процессов на уровне компаний способна привести к формированиям единых наборов компетенций, требуемых в процессе разработки, что с большой вероятностью положительно скажется на уровне квалификации специалистов за счёт повышения переиспользуемых знаний и снижения набора специфичных компетенций.

Заключение

Все описанные этапы методологии носят рекомендательный характер и не требуют безукоризненного повторения описанных практик. Этот свод правил может лежать вместе с исходным кодом проекта и содержать в себе описание всех популярных практик и операций по работе с конкретной кодовой базой, а может даже с определённым участком кода. Методология CODE может выполнять роль документации, при условии, что она станет общепринятым стандартом или приобретет поддержку общественности.

Примечание от авторов:

Возможно, мы с вами застанем время, когда новый разработчик, придя на проект, спросит: «Где посмотреть ваш CODE?».

Литература

1. P. Naur and B. Randell, (Eds.). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO (1969). URL:

https://web.archive.org/web/20191126212134/http:/homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
2. B. Randell and J.N. Buxton, (Eds.). Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969, Brussels, Scientific Affairs Division, NATO (1970). URL: https://web.archive.org/web/20191122044257/http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF
3. ГОСТ Р 57193-2016 Системная и программная инженерия. Процессы жизненного цикла систем.
4. ISO/IEC/IEEE 15288:2015. Systems and software engineering — System life cycle processes.
5. The 2020 Scrum Guide. URL: https://scrumguides.org/scrum-guide.html
6. ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.
7. Manifesto for Agile Software Development URL: https://agilemanifesto.org/
8. Sutherland, Jeff; Sutherland, J. J. Scrum: The Art of Doing Twice the Work in Half the Time. Crown Publishing Group, 2014.
9. The Testing Pyramid: How to Structure Your Test Suite. URL: https://semaphoreci.com/blog/testing-pyramid
10. Кузнецова, А. В. Проблемы информации и энтропии в медиатексте: специальность 10.01.10 "Журналистика": диссертация на соискание ученой степени кандидата филологических наук / Кузнецова Александра Владимировна. Ростов-на-Дону, 2012. 229 с.
11. Feathers, Michael C. Working effectively with legacy code. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005. 422 p.
12. Legacy system. URL: https://en.wikipedia.org/wiki/Legacy_code
13. DevOps tech: Trunk-based development. URL: https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development
14. Paul Hammant. Trunk-Based Development and Branch by Abstraction. URL: https://trunkbaseddevelopment.com/
15. B. W. Kernighan and P. J. Plauger, The Elements of Programming Style 2nd Edition, McGraw Hill, New York, 1978. 168 p.
16. Coding conventions. URL: https://kotlinlang.org/docs/coding-conventions.html
17. PEP 8 – Style Guide for Python Code. URL: https://peps.python.org/pep-0008/
18. Baum, Tobias; Liskin, Olga; Niklas, Kai; Schneider, Kurt (2016). "A Faceted Classification Scheme for Change-Based Industrial Code Review Processes". 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 74–85. doi:10.1109/QRS.2016.19.
19. Trending repositories on GitHub this month. URL: https://github.com/trending?since=monthly
20. Deming, W. Edwards (2000). Out of the crisis (1. MIT Press ed.). Cambridge, Mass.: MIT Press, 2000. p. 88. ISBN 0262541157.
21. ГОСТ Р ИСО 9001-2015. Системы менеджмента качества.
22. Hotfix. URL: https://en.wikipedia.org/wiki/Hotfix
23. Sward, David. Measuring the Business Value of Information Technology Practical Strategies for IT and Business Managers. Intell Press. 2006. 282 p.
24. «Цифровая экономика РФ»: Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации URL: https://digital.gov.ru/ru/activity/directions/858/
25. В Сбере раскрыли количество специалистов, занимающихся развитием ИИ. URL: https://www.gazeta.ru/tech/news/2022/06/14/17927438.shtml
26. Утечка исходных кодов сервисов Яндекс. URL: https://habr.com/ru/news/712888/
27. Зекирьяев Р.Т. Архитектура информационной системы быстрых сообщений под управлением протокола двунаправленного соединения и сериализации структурированных данных в сфере финансовых технологий: выпускная квалификационная работа магистра / Зекирьяев Руслан Тимурович. Москва, 2023. 97 с.

References in Cyrillics

1. P. Naur and B. Randell, (Eds.). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO (1969). Available from URL: https://web.archive.org/web/20191126212134/http:/homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
2. B. Randell and J.N. Buxton, (Eds.). Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969, Brussels, Scientific Affairs Division, NATO (1970). Available from URL: https://web.archive.org/web/20191122044257/http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF

3. GOST R 57193-2016 Sistemnaya i programmnaya inzheneriya. Processy` zhiznennogo cikla sistem.
4. ISO/IEC/IEEE 15288:2015. Systems and software engineering — System life cycle processes.
5. The 2020 Scrum Guide. Available from URL: https://scrumguides.org/scrum-guide.html
6. GOST 34.601-90. Informacionnaya texnologiya. Kompleks standartov na avtomatizirovanny`e sistemy`. Avtomatizirovanny`e sistemy`. Stadii sozdaniya.
7. Manifesto for Agile Software Development Available from URL: https://agilemanifesto.org/
8. Sutherland, Jeff; Sutherland, J. J. Scrum: The Art of Doing Twice the Work in Half the Time. Crown Publishing Group, 2014.
9. The Testing Pyramid: How to Structure Your Test Suite. Available from URL: https://semaphoreci.com/blog/testing-pyramid
10. Kuzneczova, A. V. Problemy` informacii i e`ntropii v mediatekste: special`nost` 10.01.10 "Zhurnalistika": dissertaciya na soiskanie uchenoj stepeni kandidata filologicheskix nauk / Kuzneczova Aleksandra Vladimirovna. Rostov-na-Donu, 2012. 229 s.
11. Feathers, Michael C. Working effectively with legacy code. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005. 422 p.
12. Legacy system. Available from URL: https://en.wikipedia.org/wiki/Legacy_code
13. DevOps tech: Trunk-based development. Available from URL: https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development
14. Paul Hammant. Trunk-Based Development and Branch by Abstraction. Available from URL: https://trunkbaseddevelopment.com/
15. B. W. Kernighan and P. J. Plauger, The Elements of Programming Style 2nd Edition, McGraw Hill, New York, 1978. 168 p.
16. Coding conventions. URL: https://kotlinlang.org/docs/coding-conventions.html
17. PEP 8 – Style Guide for Python Code. URL: https://peps.python.org/pep-0008/
18. Baum, Tobias; Liskin, Olga; Niklas, Kai; Schneider, Kurt (2016). "A Faceted Classification Scheme for Change-Based Industrial Code Review Processes". 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 74–85. doi:10.1109/QRS.2016.19.
19. Trending repositories on GitHub this month. URL: https://github.com/trending?since=monthly
20. Deming, W. Edwards (2000). Out of the crisis (1. MIT Press ed.). Cambridge, Mass.: MIT Press, 2000. p. 88. ISBN 0262541157.
21. GOST R ISO 9001-2015. Sistemy` menedzhmenta kachestva.
22. Hotfix. Available from URL: https://en.wikipedia.org/wiki/Hotfix
23. Sward, David. Measuring the Business Value of Information Technology Practical Strategies for IT and Business Managers. Intell Press. 2006. 282 p.
24. «Cifrovaya e`konomika RF»: Ministerstvo cifrovogo razvitiya, svyazi i massovy`x kommunikacij Rossijskoj Federacii URL: https://digital.gov.ru/ru/activity/directions/858/
25. V Sbere raskry`li kolichestvo specialistov, zanimayushhixsya razvitiem II. URL: https://www.gazeta.ru/tech/news/2022/06/14/17927438.shtml
26. Utechka isxodny`x kodov servisov Yandex. URL: https://habr.com/ru/news/712888/
27. Zekir'yaev R.T. Arhitektura informacionnoj sistemy bystryh soobshchenij pod upravleniem protokola dvunapravlennogo soedineniya i serializacii strukturirovannyh dannyh v sfere finansovyh tekhnologij: vypusknaya kvalifikacionnaya rabota magistra / Zekir'yaev Ruslan Timurovich. Moskva, 2023. 97 s.

Зекирьяев Руслан Тимурович, аспирант кафедры инструментального и прикладного программного обеспечения Института информационных технологий ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: auriax@ya.ru ORCID: 0000-0003-3787-7511

Болбаков Роман Геннадьевич, к.т.н., доцент, заведующий кафедрой инструментального и прикладного программного обеспечения Института информационных технологий ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: bolbakov@mirea.ru. ORCID: 0000-0002-4922-7260

Ключевые слова

Разработка программного обеспечения, управление разработкой программного обеспечения, кодовая база, стандарты информационных технологий.

Ruslan T. Zekiriaev, Roman G. Bolbakov. Methodology of code base management of program complexes in the conditions of digital economy

Keywords

Software Development, Software Project Management, Code Base, Information Technology Standards.

DOI: 10.34706/DE-2024-01-09

JEL classification: O32

Abstract

IT project management, along with project management in industries with a longer history, such as metallurgy, mechanical engineering, heavy industry, is a regulated process in which all stages are regulated by various documents - methodologies, agreements, standards. These regulations help all participants in the process to use practices that have proven their effectiveness – starting from the stages of project planning, ending with testing and putting the project into operation. As various processes become established, new methodologies appear that absorb all the accumulated experience and make it possible to use this experience. At the moment, there has been the formation of another series of development stage processes, the description and formalization of which makes it possible for a large number of developers to use the best practices and not reinvent the wheel in working with the code base.

This paper discusses the history of methodologies in the field of information technology and ways to formalize the practices used. Then, a brief characterization of the state of the field in the context of working with the source code of a software product is given. The next step is to describe the proven practices of working with the code base, and on their basis to systematize the accumulated knowledge and bring it to the form of a methodology ready for practical application. An example of using the methodology described in the paper serves to confirm its viability and to mark the limits of its practical application.