DOI 10.37882/2223–2966.2022.12.31

СЕМАНТИЧЕСКОЕ МОДЕЛИРОВАНИЕ: ОБЗОР ПРОЦЕССОВ, ИНСТРУМЕНТОВ, МЕТОДОВ И ЗНАНИЙ ПРЕДМЕТНОЙ ОБЛАСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ЧАСТЬ 1)

SEMANTIC MODELING OF THE SOFTWARE DEVELOPMENT DOMAIN: TOOLS, METHODS, KNOWLEDGE (PART 1)

A. Timofeev
I. Evdokimova
N. Khaptakhaeva
A. Senotrusova

Summary. The article provides an overview of the methods and tools used in the software life cycle processes, according to ISO / IEC12207 related to the categories of project processes. The review is made in order to highlight the main entities and relationships used in the semantic modeling of the subject area of software development.

Keywords: semantic modeling, systems development life cycle, software development.

Тимофеев Александр Николаевич
Генеральный директор ООО «СибДиджитал»
tan@sibdigital.net
Евдокимова Инга Сергеевна
Канд. техн. наук, доц., Восточно-Сибирский государственный университет технологий и управления
evdinga@gmail.com
Хаптахаева Наталья Баясхалановна
Канд. техн. наук, доц., Восточно-Сибирский государственный университет технологий и управления
khapnb@gmail.com
Сенотрусова Анастасия Александровна
Восточно-Сибирский государственный университет технологий и управления
saa@sibdigital.net

Аннотация. В статье проведен обзор методов и инструментов, используемых в процессах жизненного цикла программного обеспечения (ПО), согласно ISO / IEC12207, относящихся к категориям процессов проекта. Обзор выполнен в целях выделения основных сущностей и отношений, используемых при семантическом моделировании предметной области разработки программного обеспечения.

Ключевые слова: семантическое моделирование, жизненный цикл программного обеспечения, разработка программного обеспечения.

Введение

В настоящее время разработка программного обеспечения ведется с применением разнообразных подходов и практик, которые как правило объединяются в группы, называемые методологиями разработки программного обеспечения (Software Development Methodology, SDM) [1]. Выделяют несколько основных методологий разработки программного обеспечения.

Каскадная модель (waterfall model), в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки. Данная модель реализована в таких стандартах, как PMBOK, ГОСТ 34.601–90.

V–Model, являющаяся вариацией каскадной модели, в которой детализация проекта возрастает при движении слева направо, одновременно с течением време-

ни. Данная модель реализована в таких стандартах, как PRINCE2, VEE, V–Model XT.

Гибкая методология (agile software development), к которой в частности, относят экстремальное программирование, DSDM, Scrum, FDD, BDD и другие.

Итеративный подход, основанный на повторяющемся цикле PDCA, к которому могут быть отнесены Rational Unified Process, OpenUP.

Спиральная модель, сочетающая итеративность и этапность.

Прочие: RAD, DevOps, Канбан, Lean.

Многообразие методологий обуславливает необходимость определения действенных критериев выбора методологии, наиболее соответствующей задачам, ресурсам и ограничениям конкретных проектов, а также накладывает необходимость постоянного анализа инструментальных средств, используемых при реализации процессов методологии.

Совокупность процессов образует жизненный цикл разработки программного обеспечения (Software Development Life Cycle, SDLC), то есть процесс, используемый индустрией программного обеспечения для проектирования, развертывания, разработки и тестирования программного обеспечение [1].

В рамках настоящей работы процессы жизненного цикла рассматриваются в соответствии с международным стандартом ISO / IEC12207 (в России — ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»), который определяет задачи, необходимые для разработки и поддержки программного обеспечения, и включает следующие категории процессов: процессы соглашения; процессы организационного обеспечения проекта; процессы проекта; технические процессы; процессы реализации программных средств; процессы поддержки программных средств; процессы повторного применения программных средств.

Целью настоящей работы является семантическое моделирование предметной области «Разработка программного обеспечения». Семантическое моделирование позволит понять основные сущности предметной области и отношения между ними. Предполагается, что будут исследованы концептуальные отношения — квалитативные и квантитативные в разрезе следующих понятийных сфер: абстрактное-конкретное, принадлежность, форма и содержание, процессуальность, тождество и противоречие. При этом потребуется распределить отношения по уровням абстракции [5].

Формирование системного понимания о текущем смысле предметной области и перспективных задачах ее развития определяет актуальность работы в условиях постоянного появления и развития различных разделов компьютерной науки.

Для достижения поставленной цели необходимо выполнить обзор методов и инструментов, используемых в процессах жизненного цикла ПО, относящихся категориям: процессы проекта, технические процессы, процессы реализации и процессы поддержки [2].

Процессы проекта

Существуют две категории процессов проекта. Процессы менеджмента проекта используются для планирования, выполнения, оценки, управления продвижением проекта, а также могут привлекаться в любое время жизненного цикла и на любом уровне иерархии проекта. Процессы поддержки проекта обеспечивают выполнение специализированных целей менеджмента [2]. Процессы данной группы не специфичны для области разработки программного обеспечения, однако их выходы определяют возможности выполнения процессов других категорий процессов ISO / IEC12207.

Процесс планирования проекта

Цель процесса планирования проекта состоит в составлении и доведении до заинтересованных сторон эффективного и выполнимого плана. Данный процесс определяет область применения менеджмента проекта и технических мероприятий, результаты процесса, проектные задачи и поставки, устанавливает графики для выполнения задач проекта, включая критерии достижения и ресурсы, необходимые для выполнения задач проекта [2].

Для распределения задач и определения графика работ используют инструменты календарно-сетевого планирования, такие как Microsoft Project (или Microsoft Project Web Application), ProjectLibre,]project-open[, а также информационные системы управления проектами, например Open Project или Oracle Primavera.

Как правило на данном этапе детальная декомпозиция работ не требуется, поэтому часто применяют такие подходы как построение диаграммы Гантта с указанием работ верхнего уровня, зависимостей между ними, сроков выполнения и ресурсов.

Работы верхнего уровня как правило переносят в системы управления задачами в качестве группирующих

сущностей, позволяющих в крупных проектах консолидировать информацию о ходе проекта. Например, в Redmine такие работы становятся подпроектами в рамках проектов или родительскими задачами, которые детализируются подзадачами, в Atlassian Jira — эпиками (Epic) или родительскими задачами. Использование иерархии задач может вызывать неожиданные эффекты при автоматическом определении статуса родительской задачи, при подсчете трудозатрат или фильтрации, а также трудности в случае изменения объема работ. Поэтому рациональным представляется использование отношения «Объект — свойство» [5] (дополнительное настраиваемое поле в Redmine или эпик в Atlassian Jira), чем «Целое-часть» [5] (подпроект в Redmine, родительские задачи в Redmine или Atlassian Jira).

Существуют различные эвристические рекомендации по составлению планов ИТ-проектов [11], или концептуальные рекомендации, сформулированные как максимы, например «если хотите определить реальные сроки ИТ-проекта, то надо имеющийся срок умножить на число π или число e».

Оценка проекта и процесс управления

Цель оценки проекта и процесса управления заключается в определении состояния проекта и гарантии того, что проект выполняется в соответствии с планами и графиками работ в пределах бюджета и удовлетворяет техническим параметрам [2]. Для определения состояния проекта используются управленческие и технические инструменты, которые помогают установить отклонение текущего состояния проекта от плана.

Среди управленческих инструментов определения состояния проекта выделяются мониторинговые инструменты: канбан-доски, реализованные в таких инструментах, как Trello, Wrike, Asana и agile-диаграммы: сгорание задач (burndown), накопительная диаграмма потока (cumulative flow), производительность (velocity), время от появления до поставки (lead time) и другие [3]. Применяются методики по декомпозиции работ проекта на спринты или итерации.

К промежуточному уровню между управленческими и техническими инструментами относят создание минимально жизнеспособного продукта (minimum viable product, MVP), то есть продукта, обладающего минимальными, но достаточными для удовлетворения первых потребителей функциями или управление техническим долгом, то есть тем, что было запланировано и тем, что было поставлено [4]. Появление технического долга позволяет ускорить доставку функциональности продукта конечному пользователю путем решения в основном задач, относящихся к обязательной сложности [4], но при этом ведет к накоплению внутренних недоработок и временных решений («костыли»).

Технические и инструменты, как правило, фокусируются на выявлении проблем в производительности отдельных разработчиков и команды в целом. Для измерения производительности вводятся различные метрики, описанные в разделе «Процесс измерений» настоящей статьи. Эффективность применения, которых до сих пор остается дискуссионной [6].

Процесс менеджмента решений

Цель процесса менеджмента решений заключается в выборе из существующих альтернатив наиболее предпочтительного направления проектных действий. Решения, принимаемые заинтересованными сторонами, и их обоснования документируются таким образом, который позволяет проводить аудит и изучать полученный опыт [2].

Перед выработкой проектных решений необходимо рассматривать существующие альтернативы и анализировать предыдущий опыт. В качестве простейших мер используют обсуждение внутри команды, анализ ранее осуществленных проектов и обязательный поиск подобных ситуаций в сети Интернет. При решении более сложных задач организуют ведение собственной корпоративной базы знаний, формы ведения которой описаны в разделе «Процесс менеджмента информации» настоящей статьи. Ведение корпоративной базы знаний является сложной и трудозатратной задачей, сама по себе база знаний может быть нематериальным активом, имеющим собственную ценность [9], и одновременно существенно снижать трудозатраты на решение многих типовых проблем [8].

Для упрощения принятия технических решений используются такие средства, как awesome-репозитории (awesome-списки), которые содержат структурированную информацию по разнообразным вопросам и позволяют сравнить альтернативы. Среди них можно выделить имеющий более 200 000 звезд GitHub-репозиторий «awesome» (URL: https://github.com/sindresorhus/awesome) или специализированные «awesome-java» (URL: https://github.com/akullpp/awesome-java) для Java или «system-design-primer» (URL: https://github.com/donnemartin/system-design-primer) с актуальными примерами архитектур.

На практике принятые решения документируются, а также по возможности сохраняется история выработки решений. Документирование решений осуществляется способами, описанными в разделе «Процесс менеджмента информации» настоящей статьи.

История выработки решений может быть сохранена по-разному. Фиксируются правки и комментарии к задачам в таск-трекере, хранятся различные версии проектных документов, при необходимости ведутся аудиозаписи конференций и интервью, а также протоколируются очные собрания.

Процесс менеджмента рисков

Цель процесса менеджмента рисков заключается в постоянном определении, анализе, обработке и мониторинге рисков [2]. Важность процесса подтверждается тем, что согласно данным The Standish Group International в 2014 году, среднее отклонение от запланированных бюджетов, сроков выполнения и качества в ИТ-проектах, разрабатываемых в западных странах, составило 89%, однако в случае разработки и внедрения организационно-методического обеспечения менеджмента рисков отклонение фактических результатов от запланированных составило 4,5% [10].

Процесс менеджмента рисков выполняется на протяжении всего жизненного цикла продукта и включает: идентификацию, анализ, состоящий из оценки вероятности и оценки влияния риска, реагирования на риски и контроля и мониторинга рисков [2].

Идентификация рисков выполняется с применением различных методов: блок-схемы принятия решений (Process Decision Program Chart, PDPC), опросов, мозговых штурмов, SWOT-анализа, проведения интервью [10].

При анализе рисков используют качественные методы анализа, например: дерево событий (Event Tree Analysis, ETA), дерево неисправностей (Fault Tree Analysis, FTA), галстук-бабочка, причинно-следственная диаграмма Исикавы (Fishbone Diagram). Для качественной оценки степени влияния и вероятности наступления рисков применяют вербально-числовую шкала Харрингтона или классификацию Т. Мерны и Ф. Тхани (T. Merna, F. Al-Thani), включающую катастрофические, непредсказуемые, частые и несущественные риски [10].

Мероприятия по реагированию на риски разрабатываются с применением методов Уолта Диснея, разделения мышления де Боно или метода Делфи [10].

На основании практического опыта можно выделить некоторые распространенные группы рисков.

Возникновение скрытых работ. Скрытые работы — это работы, не включенные в план проекта. Для избежания возникновения данного риска на начальных этапах проводят сравнения с аналогами, прототипирование, детализируют требования, выполняют декомпозицию задач «сверху вниз» до уровня, позволяющего достаточно легко реализовать подзадачу выбранными средствами. При реагировании на такие риски проводят приоритезацию задач на блокирующие достижение результата и такие задачи, которые могут быть реализованы частично, в виде MVP или отложены. При этом необходимо учитывать, что в таком случае возникает «технический долг» [4].

Конфликты версий. Данный риск проявляется в виде так называемого «ада зависимостей» (Dependency hell) или конфликтов в системах управления версиями [14]. Классические формы ада зависимостей — множество зависимостей, длинные цепочки, конфликтующие и циклические зависимости — достаточно широко описаны, как и подходы к решению данной проблемы. Однако, даже в слабосвязанной микросервисной архитектуре возникают ситуации, конфликтов версий API различных микросервисов. Для избежания этого проводят мониторинг и аудит зависимостей проекта, планируют выпуск версий компонентов и их интеграционное тестирование. Для снижения трудозатрат и рисков ошибочного разрешения конфликтов используют рабочие процессы, например, Git-flow [15], декомпозируют задачи на сопоставимые объемы, связывают их с ветками репозитория и контролируют сроки «жизни» веток: чем дольше ветка кода развивается параллельно основной, тем сложнее будет произвести их слияние. Более сложным случаем потенциальных конфликтов версий является ведение нескольких версий продукта или микросервиса.

Конфликты конфигурации окружения. Окружение — это компьютерная система, в которой программа или ее компонент разворачивается и выполняется. Выделяют различные классификации окружений, например: development (DEV) и production (PROD); каскад окружений deployment, testing, model, production (DEV, TEST, MODL, PROD); каскад окружений deployment, testing, acceptance, production (DTAP); Quality Control (QC) и экспериментальное окружение (EXP); окружение пользователя (USER) или локальное окружение (LOCAL). Множественность окружений — это естественное состояние для программного обеспечения, избежать данные риски невозможно, вероятность и негативный эффект от их наступления, снижается с помощью подхода «конфигурация как код» (Configuration-as-code, CaC, GitOpts)[12].

Лицензионные риски. Данные риски подразделяются на связанные с лицензированием стороннего программного обеспечения, применяемого в проекте и связанные с передачей лицензии на создаваемое программное обеспечение.

Риски первой группы связаны с влиянием лицензии компонента на все составное произведение (например, если компонент лицензирован под лицензией GNU GPL),

юрисдикцией, в которой создается программное обеспечение (например, в России существует позиция о неприменимости ограничений лицензий GPL к производным в результате переработки произведениям [17]). Условия лицензирования отличаются в зависимости от целей использования как было с некоторыми версиями Oracle Open JDK [18] или меняются со временем, например, у Elasticsearch открытая лицензия Apache 2.0 была изменена на собственную. В настоящее время возросли риски ограничений лицензирования и поддержки в зависимости от территориальной принадлежности, например, такие ограничения налагают дистрибутивы Linux CentOS и Fedora.

К рискам второго типа относятся риски лицензирования создаваемого продукта. Например, с 2021 года действует Приказ Минцифры России от 17.12.2020 № 715, требующий при исполнении госконтрактов передавать исключительные права и исходный код созданной системы. На практике возникают ситуации, когда передаются права на составное произведение: неисключительная лицензия на платформенную часть, и исключительная на выполненные доработки. В этих случаях процесс сборки программного обеспечения (описать и воспроизвести который требует Приказ № 715) изменяется от инсталляции к инсталляции.

Риски возникновения уязвимостей. Под уязвимостями следует понимать недостатки в системе, используя которые можно намеренно нарушить её целостность и вызвать неправильную работу. Учитывая сложность современного программного обеспечения, данные риски являются практически неизбежными. Для снижения этих рисков обновляют зависимости с учетом таких источников информации, как Common Vulnerabilities and Exposures (CVE), National Vulnerability Database (NVD), Common Weakness Enumeration (CWE), Vulners, VulDB, ICS-CERT, Snyk, Банка данных угроз ФСТЭК, применяют стандарты Common Vulnerability Scoring System (CVSS) или ethicsfIRST [19] и автоматизированные инструменты, например dependabot для GitHub или GitLab[19]; проводят статический анализ кода с помощью таких инструментов, как SonarQube, PMD, ReSharper, JaCoCo, Codacy, FindBugs, PVS-Studio; выполняют различные виды тестирования программного обеспечения.

Для снижения вероятности и негативных эффектов от рисков, связанных с техническими процессами, такими как проектирование и кодирование можно рекомендовать по возможности использовать шаблоны проектирования и избегать или выявлять и устранять антипаттерны.

Процесс менеджмента конфигурации

Цель процесса менеджмента конфигурации состоит в установлении и поддержании целостности всех идентифицированных выходных результатов проекта или процесса обеспечения доступа к ним любой заинтересованной стороны [2].

В рамках данного процесса в проекте должна быть определена стратегия менеджмента конфигурации и идентифицированы предметы управления конфигурацией [2]. Управление конфигурацией тесно связано с инструментами непрерывной интеграции (Continuous Integration), такими как GitLab CI, Github Actions, Jenkins, TeamCity

В настоящее время одним из наиболее популярных подходов к управлению конфигурацией является подход «конфигурация как код» (Configuration-as-code, CaC, GitOpts) [12], данный подход постулирует, что git-репозитории с конфигурацией являются единственным доверенным источником и содержат декларативно описанное состояние приложения, ожидаемое в конкретном окружении. Конфигурация как код предназначена для управления параметрами конфигурации конкретного приложения, которые отделены от кода инфраструктуры и управляются в рамках собственного процесса [12]. Среди инструментов, позволяющих реализовать данный подход, можно выделить ArgoCD, Flux, Tekton, Werf [13], а также связанные инструменты, обеспечивающие подход Инфраструктура как код (IaC) [12]: Terraform, SaltStack, Puppet, Chef.

При управлении конфигурацией используют следующие практики.

Применяют для управления конфигурацией общепринятые форматы YAML, JSON, XML, PROPERTIES и подходы, например, разделение на профили и среды, создание скриптов сборки и развертывания.

Распределяют конфигурации по средам на среду разработки, тестовую, продуктивную и др.

Выносят конфигурации в источники данных, доступные для изменения без необходимости повторной сборки приложения или перезапуска.

Процесс менеджмента информации

Цель процесса менеджмента информации состоит в своевременном предоставлении заинтересованным сторонам релевантной, своевременной, полной, достоверной информации в течение жизненного цикла системы. В рамках данного процесса реализуется создание, сбор, преобразование, хранение, поиск, распространение и использование информации [2].

Управление технической, проектной и пользовательской информацией может осуществляться следующими способами.

В виде wiki-страниц в таких системах как Redmine, OpenProject, полноценных wiki-движков, например, MediaWiki или DokuWiki.

В виде баз знаний, построенных как систематизированные наборы файлов, страниц или заметок OneNote или ведущихся в специализированных системах управления знаниями (Knowledge Management Systems, KMS), таких как Atlassian Confluence, Notion, Obsidian, Helpy или Documize, Gollum.

Путем хранения документов в общем доступе с помощью систем управления контентом (ECM-систем), таких как NextCloud, OwnCloud, Alfresco, SharePoint или облачных хранилищ Google Docs, Облако mail.ru.

Непосредственно в исходном коде проекта в виде markdown-файлов. Общепринятым способом является размещение файла Readme.md в git-репозитории проекта.

Проектную, организационную и информацию из соглашений хранят в системах управления контентом, например, NextCloud, OwnCloud, Alfresco, SharePoint, при этом целесообразно использовать облачные хранилища или сетевые каталоги с заданной структурой.

Процесс измерений

Цель процесса измерений заключается в сборе, анализе и составлении отчетов о данных, относящихся к разработанным продуктам и процессам, реализованным в пределах определенного организационного подразделения, для поддержки эффективного менеджмента процессов и объективной демонстрации качества этих продуктов [2].

В рамках данного процесса идентифицируется, что и в каких единицах необходимо измерять. Как правило, измерить можно параметры процесса и параметры продукта.

В качестве метрик процесса часто используется простейшая информация об активности в GitLab или GitHub с указанием количества пушей (push) в репозитории, принятых merge request, комментариев и т.д., информация о количестве закрытых задач в Jira, Redmine или ином трекере задач.

Могут использоваться различные отчеты agile-методологии: сгорание задач (burndown), накопительная диаграмма потока (cumulative flow), производительность (velocity), время от появления до поставки (lead time) и другие [3].

Измерение параметров продукта как правило заключается в расчете определенных метрик, например: покрытия требований, покрытия кода, цикломатической сложности. Существуют системы метрик, включающие отношение созданных и закрытых дефектов, состава релизов, коэффициент ошибок, время жизни дефектов и т.д. [6]. Для работы с данными метриками может быть использован такой инструмент как TeamMeter [6].

Указанные метрики, как правило, фиксируют текущую ситуацию и дают возможность в некоторых случаях проводить оценки «по аналогу». Для анализа причин и выявления неочевидных закономерностей используют сочетания метрик и паттерны, основанные на анализе этих метрик [7]. Для этого могут быть применены такие инструменты как CodeScene, Monocle, CodeClimate Velocity, Waydev, Pluralsight Flow, LinearB, GitClear [7].

Качество кода оценивается с помощью статических или динамических анализаторов, анализа производительности или различных видов тестирования [16]: модульного, интеграционного, функционального нагрузочного и т.д.

Заключение

В настоящей статье описаны основные модели разработки программного обеспечения и основные процессы, дан краткий обзор инструментов, а также некоторых методов и практических приемов применяемых для решения задач, относящихся к категории процессов проекта. Подробно описаны методы и приемы работы с рисками.

Таким образом, модели разработки программного обеспечения являются родовым понятием по отношению к различным стандартам и методологиям (абстрактное-конкретное, отношение иерархии), стандартны и методологии находятся в отношении корреляции (тождество и противопоставление). Процессы являются частью стандартов (принадлежность, отношение агрегации), вместе с тем, процессы разных стандартов могут коррелировать, или быть синонимами (тождество и противопоставление).

Процессы включают классы задач и методы решения (принадлежность, отношение агрегации), которые имплементируются инструментами (объект-действие-субъект, процессуальность), то есть классами систем и конкретными системами (абстрактное-конкретное, отношение иерархии «Род-Вид»). Для категории процессов проектов можно выделить следующие основные виды систем: системы управления проектами и задачами; системы расчета метрик; системы управления знаниями;

системы управления контентом; системы управления конфигурациями.

Системы имеют свойства и применимы в конкретной среде (принадлежность, объект-свойство, объект-пространство реализации). Классы решаемых задач и формы их решения в конкретных системах описываются семантическими отношениями «Термин-способ выражения» и «Термин-способ представления».

Инструменты и методы, применяемые в процессах, описываются функциональными отношениями: процесс — действие, инструмент — применение, риск — действие, риск — условие возникновения, метрики/данные — действия, метрики/данные — величины.

Таким образом, анализ категории процессов проектов приводит к выявлению ряда объектов и основных отношений между ними.

Статья является первой части работы по анализу процессов жизненного цикла программного обеспечения. В следующей части предполагается провести обзор процессов категории «Технические процессы».

ЛИТЕРАТУРА

1. Перл И.А. Введение в методологию программной инженерии / И.А. Перл О.В. Калёнова — СПб, Изд-во ИТМО, 2019–53 с.
2. ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»
3. Метрики в Scrum и Kanban [Электронный ресурс]. URL: https://habr.com/ru/post/173565/ (Дата обращения: 05.10.2022).
4. Technical Debt [Электронный ресурс]. URL: https://martinfowler.com/bliki/TechnicalDebt.html (Дата обращения: 05.10.2022).
5. Найханова Л.В. Технология создания методов автоматического построения онтологий с применением генетического и автоматного программирования [Текст] / Л.В. Найханова.— Улан-Удэ: Издательство БНЦ СО РАН, 2008.— 287 с.
6. Сказ про то, как мы метрики качества внедряли [Электронный ресурс]. URL: https://habr.com/ru/company/tinkoff/blog/684608/ (Дата обращения: 05.10.2022).
7. Покажи мне свой Git, и я скажу, кто ты [Электронный ресурс]. URL: https://habr.com/ru/company/oleg-bunin/blog/691468/ (Дата обращения: 05.10.2022).
8. Холистическое управление знаниями в IT-компании [Электронный ресурс]. URL: https://habr.com/ru/company/oleg-bunin/blog/491884/ (Дата обращения: 08.10.2022).
9. Гаврилова, Т.А. Управление знаниями с российским акцентом: победы и поражения / Т.А. Гаврилова, А.И. Алсуфьев, Л.О. Кокоулина // Инновации.— 2017.— № 1(219).— С. 59–69.— EDN YLZLOJ.
10. Николаенко, В.С. Внедрение риск-менеджмента в ИТ-проекты / В.С. Николаенко // Государственное управление. Электронный вестник.— 2016.— № 54.— С. 63–88.— EDN VODSNF.
11. Об оценках сроков в разработке ПО [Электронный ресурс]. URL: https://habr.com/ru/post/506486/ (Дата обращения: 10.10.2022).
12. Config as Code: What is it and how is it beneficial? [Электронный ресурс]. URL: https://octopus.com/blog/config-as-code-what-is-it-how-is-it-beneficial (Дата обращения: 10.10.2022).
13. GitOps — что это такое и с чем его едят? [Электронный ресурс]. URL: https://habr.com/ru/company/oleg-bunin/blog/690544/ (Дата обращения: 10.10.2022).
14. Merging [Электронный ресурс]. URL: https://web.mit.edu/gnu/doc/html/cvs_9.html (Дата обращения: 10.10.2022).
15. Рабочий процесс Gitflow Workflow [Электронный ресурс]. URL: https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflow-workflow (Дата обращения: 11.10.2022).
16. Теория тестирования ПО просто и понятно [Электронный ресурс]. URL: https://habr.com/ru/post/587620/ (Дата обращения: 12.10.2022).
17. Близнец И.А. Интеллектуальная собственность в современном мире: монография / под ред. И.А. Близнеца.— М.: Проспект,— 2016.— 672 с.
18. Introducing the Free Java License [Электронный ресурс]. URL: https://blogs.oracle.com/java/post/free-java-license (Дата обращения: 12.10.2022).
19. Что такое CVE и какие угрозы там хранятся? [Электронный ресурс]. URL: https://habr.com/ru/company/pvs-studio/blog/678410/ (Дата обращения: 12.10.2022).