DOI 10.37882/2223–2966.2023.04–2.30

СЕМАНТИЧЕСКОЕ МОДЕЛИРОВАНИЕ: ОБЗОР ПРОЦЕССОВ, ИНСТРУМЕНТОВ, МЕТОДОВ И ЗНАНИЙ ПРЕДМЕТНОЙ ОБЛАСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ЧАСТЬ 2)

SEMANTIC MODELING OF THE SOFTWARE DEVELOPMENT DOMAIN: TOOLS, METHODS, KNOWLEDGE (PART 2)

A. Timofeev
I. Evdokimova
N. Khaptakhaeva

Summary. The article provides an overview of the methods and tools used in the software life cycle processes, according to ISO / IEC 12207 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, доцент, Восточно-Сибирский государственный университет технологий и управления
evdinga@gmail.com
Хаптахаева Наталья Баясхалановна
Канд. техн. наук, доцент, Восточно-Сибирский государственный университет технологий и управления
khapnb@gmail.com

Аннотация. В статье проведен обзор методов и инструментов, используемых в процессах жизненного цикла программного обеспечения (ПО), согласно ISO / IEC 12207, 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 рамках настоящей работы рассматриваются процессы жизненного цикла в соответствии с международным стандартом ISO / IEC 12207 (в России — ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств») .

Поскольку исследование требует обработки слабоструктурированных текстовых данных, то для построения семантической модели может быть применен подход, называемый «слоеный пирог обучения онтологий» [14], предполагающий последовательное структурирование информации в порядке, показанном на рисунке 1.

В рамках выбранного стандарта при анализе материалов выделяются термины (инструменты, системы), устанавливаются отношения синонимии между ними (одинаковая функциональность), выделяются концепты (классы систем и инструментов) и отношения между ними уже установлена [2]. Термины, концепты и отношения между ними могут быть определены вне стандартов. Описания процессов в каждом стандарте можно считать аксиомами. Учитывая множественность стандартов отношения с применением указанного подхода, могут быть установлены и между стандартами и их составными частями.

В первой части были рассмотрена процессы, относящиеся к группе «Процессы проекта». В результате обзора содержательной части данных процессов был выявлен ряд классов сущностей и семантических отношений между ними [9] .

Во второй части рассматриваются процессы группы «Технические процессы» и будут исследованы концептуальные отношения — квалитативные и квантитативные в разрезе следующих понятийных сфер: абстрактное-конкретное, принадлежность, форма и содержание, процессуальность, тождество и противоречие.

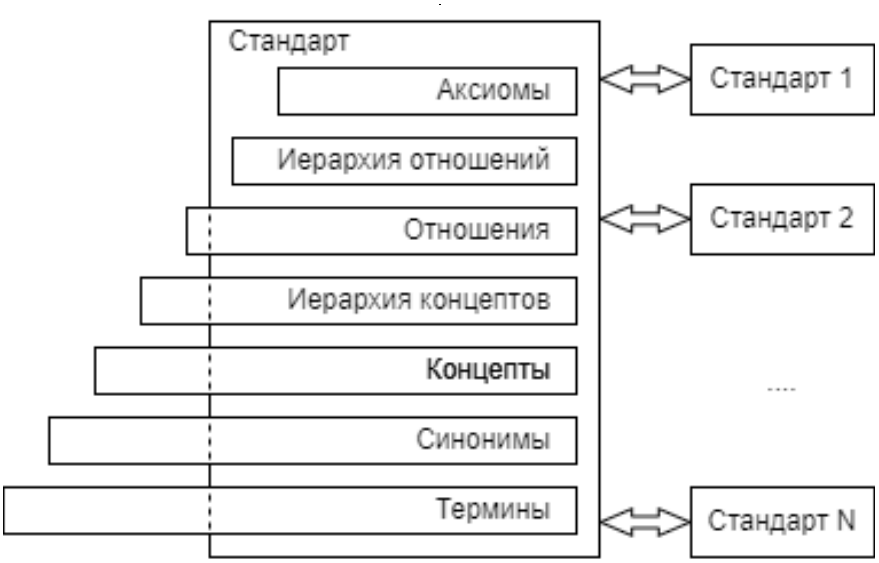

Рис. 1. «Подход к построению семантической модели»

Технические процессы

Технические процессы используются для определения требований, их преобразования в полезный продукт, применения продукта, обеспечения требуемых услуг, поддержания обеспечения этих услуг и изъятия продукта из обращения, если он не используется при оказании услуги. Технические процессы определяют деятельность, которая дает возможность реализовывать организационные и проектные функции, являющиеся следствием технических решений и действий, и определяет такие свойства продуктов, как своевременность, доступность, результативность затрат, а также функциональность, безотказность, ремонтопригодность, продуктивность, приспособленность к применению и т.д. [1] .

Процесс определения требований правообладателей

Цель процесса определения требований правообладателей — выявить требования, выполнение которых может обеспечивать предоставление услуг, необходимых пользователям и другим правообладателям в заданной среде применения [1].

Требования правообладателей могут выражаться в форме потребностей, пожеланий, требований, ожиданий и воспринятых ограничений отдельных правообладателей, которые, в свою очередь, выражаются в терминах текстовой или формализованной модели [1]. Для формализации требований на данном этапе могут быть использованы специальные языки спецификаций и нотации, такие как UML (Unified Modeling Language), ERD (Entity-Relationship Diagrams), DFD (Data Flow Diagrams), RDL (Report Definition Language), BPMN (Business Process Modeling Notation), семейство IDEF (Integrated Definition), ARIS (Architecture of Integrated Information Systems), Mind map и др., которые позволяют описать и структурировать требования к создаваемой системе, сопоставить и установить связи требований с источниками, а также сделать требования более понятными и доступными для всех участников процесса разработки.

Среди инструментов для работы с диаграммами вариантов использования (use case) можно выделить инструменты, ориентированные на визуализацию — Dia, Draw.io, Lucid Chart и ориентированные на структуризацию — Microsoft Visio.

Среди инструментов для работы с ментальными картами можно выделить FreeMind (интегрируется с wiki-движками), Docear, Microsoft Visio, MindManager XMind, Miro.

Необходимо учитывать, что в рамках данного процесса определяется базовая версия требований, которая в дальнейшем может существенно изменяться. Для того, чтобы изменения не были стихийными необходимо определить правила составления версий требований, статусы требований, способы внесения изменений, методы анализа влияния предлагаемых изменений и способы отслеживания связей плана проекта, внешних факторов и текущей версии требований.

Процесс анализа системных требований

Цель анализа системных требований состоит в преобразовании определенных требований правообладателей в совокупность необходимых системных технических требований — одинаково понимаемых всеми заинтересованными сторонами утверждений о свойствах системы — которыми будут руководствоваться в проекте системы. Также в процессе анализа требований необходимо убедиться в полноте, точности и достаточности требований, что связано с задачей оценки качества требований.

Не существует универсальной модели оценки качества требований. Среди имеющихся можно выделить: ISO/IEC/IEEE 29148:2018 «Systems and software engineering — Life cycle processes — Requirements engineering» [10], BABOK Guide v3.0 (Business Analysis Body of Knowledge), CMMI (Capability Maturity Model Integration), ГОСТ Р ИСО/МЭК 25051–2017 [11] и ГОСТ Р ИСО/МЭК 25010–2015 [12] и др. Каждый из них содержит свои собственные методы и меры оценки требований.

Однако, анализируя их, можно выделить характерные критерии: прослеживаемость между системным требованием и базовой линией требований заказчика, корректность с технической и юридической точек зрения, согласованность с потребностями заказчика и другими требованиями, полнота описания, модульность, тестируемость и идентифицируемость, осуществимость с точки зрения архитектуры, осуществимость функционирования и сопровождения [3] и др.

При этом выделяются такие виды требований, как функциональные, управленческие, эргономические, архитектурные, а также требования к взаимодействию и сервисного уровня.

В целях соответствия указанным критериям при управлении требованиями выделяют ряд составляющих: управление версиями, управление изменениями, отслеживание состояния требований, отслеживание связей требований [4], которые обеспечиваются системами управления требованиями (СУТ), Requirements Management Systems, RMS. Среди них можно выделить IBM Rational ClearQuest, Doorstop, rmToo, SILA Union, REQCHECKER, Micro Focus Dimensions RM, ReqView, Case Complete и др.

На практике многие проекты начинают реализовываться в условиях, когда не все требования сформулированы, что обуславливает необходимость управления «отложенными решениями». Риски, связанные с «отложенными решениями», как правило коррелируют со степенью соответствия требований критериям, описанным выше. Управление рисками, связанными с «отложенными решениями», требует комплексного подхода и постоянного мониторинга, а расстановка приоритетов и разработка стратегии управления рисками являются ключевыми элементами этого процесса [13] .

Процесс проектирования архитектуры системы

Цель процесса проектирования архитектуры системы заключается в определении того, как системные требования следует распределить относительно элементов системы [1]. Для этого должна быть создана и оценена архитектура, определяющая основные компоненты системы, внутренние и внешние интерфейсы взаимодействия. Функциональные и нефункциональные требования должны быть распределены по компонентам системы таким образом, чтобы прослеживалась взаимосвязь с базовой линией заказчика.

В рамках проектирования архитектуры системы создается логическая архитектура, состоящая из набора связанных технических концепций и принципов, которые можно охарактеризовать как набор решений, существенно не изменяющихся при изменении бизнес-процессов, но существенно влияющих на совокупную стоимость владения системой (Total Cost of Ownership, TCO). Поэтому при создании информационных систем следует стремиться выбирать архитектуру системы с минимальной совокупной стоимостью владения, состоящей из плановых затрат и стоимости рисков [5] .

Выделяют следующие типы архитектур: бизнес-архитектура (business architecture), ИТ архитектура (information technology architecture), архитектура данных (data architecture), а также архитектура знаний (knowledge architecture), архитектура приложения (application architecture) или программная архитектура (software architecture), техническая архитектура (hardware architecture) [5]. В зависимости от функциональности и сложности системы могут быть рассмотрены архитектура безопасности (security architecture), архитектура облачных вычислений (cloud architecture) и архитектура интернета вещей (IoT architecture).

Детальное изложение подхода к описанию архитектуры приводится в ГОСТ Р 57100–2016 Описание архитектуры, соответствующем международному стандарту ISO/IEC/IEEE 42010:2011 «Systems and software engineering — Architecture description». В нем описаны семантические отношения контекста описания архи-

тектуры, концептуальной модели описания архитектуры, элементов, связей, решений, структуры, языка описания архитектуры и других сущностей и связей между ними [6]. Например, система представляется в архитектуре (семиотическое отношение «термин — способ представления»), а архитектура выражается в описании архитектуры (семиотическое отношение «термин — способ выражения»), которое определяет рассматриваемую систему (семиотическое отношение «термин — способ выражения») [2] .

Важным аспектом проектирования архитектуры является выбор нотаций и методологий, с помощью которых осуществляется моделирование, среди которых распространены следующие: IDEF — семейство структурных моделей и соответствующих им диаграмм; DFD — диаграммы потоков данных; ERD — диаграммы «сущность-связь»; Workflow — технология управления потоками работ; BPMN; раскрашенные сети Петри (Color Petri Nets, CPN); унифицированный язык моделирования UML и его расширение SysML (The Systems Modeling Language); интегрированные средства и методологии широкого назначения, например ARIS (Architecture of Integrated Information Systems), SADT (structured analysis and design technique) или TOGAF (The Open Group Architecture Framework), использующая нотацию ArchiMate (Architecture-Animate) [5]. Также популярными являются методологии Lean Architecture и Agile Architecture основанные на принципах Lean Thinking и Agile-разработки и использующие SAFe (Scaled Agile Framework) для масштабирования Agile-разработки на предприятии.

Для моделирования в перечисленных выше нотациях применяются такие инструменты как Microsoft Visio, Dia, Draw.io, Lucid Chart, Archi, Camunda Modeler, StarUML, Bizagi Modeler, PlantUML, Mermaid, UMLet, bpmn-js, ERD.

При проектировании архитектуры распространена практика обращения к стандартам, лучшим практикам, стилям, готовым моделям, шаблонам и паттернам проектирования, таким как MVC (Model-View-Controller), Flux, MVP (Model-View-Presenter), PAC (Presentation–abstraction–control), MVVM (Model-View-ViewModel), GERAM (Generalised Enterprise Reference Architecture and Methodology), RM-ODP, Схема Захмана, модель представления Крухтена «4+1». Также могут использоваться сборники примеров, такие как https://github.com/donnemartin/system-design-primer.

Процесс комплексирования системы

Цель процесса комплексирования системы заключается в объединении системных элементов для создания полной системы, которая будет удовлетворять проекту и ожиданиям заказчика [1] .

В результате данного процесса составные части системы должны быть сконфигурированы и объединены в единый готовый для тестирования комплекс. Готовность к тестированию определяется наличием документированных тестов для каждого квалифицированного требования к системе.

Успешность комплексирования системы зависит от того, насколько эффективно проведены анализ требований и проектирование системы с точки зрения обеспечения высокой производительности, надежности и безопасности, а также от следующих этапов:

- декомпозиции системы на подсистемы, компоненты, сервисы и модули. Для этого могут использоваться хорошо зарекомендовавшие себя нотации и средства моделирования, примеры которых описаны в предыдущем разделе;
- выделения зависимостей между объектами декомпозиции. Данный этап может быть успешно выполнен путем применения пяти архитектурных принципов SOLID (SRP — принцип единственной ответственности, single responsibility principle; OCP — принцип открытости/закрытости, open-closed principle; LSP — принцип подстановки Лисков, Liskov substitution principle; ISP — принцип разделения интерфейса, interface segregation principle; DIP — принцип инверсии зависимостей, dependency inversion principle) [7];
- определения программных интерфейсов (API — Application Programming Interface). Правильное определение программных интерфейсов важно, поскольку с их использованием компоненты системы взаимодействуют друг с другом. Один из критериев правильности может быть описан как соответствие сигнатуры функции ее семантике. Для этого важно использовать средства документирования API, такие как Redoc, Swagger UI, OpenAPI-GUI, Apicurio, stoplightio, RAML и др.;
- определения конкретных реализаций компонентов. Для этого должно быть принято решение о том, будет ли использован готовый компонент или необходима разработка нового. В случае разработки нового должен быть определен технологический стек. При выполнении данной работы могут быть использованы своды лучших практик и awesome-списки (Awesome Python, Awesome React, Awesome CSS и пр.), поскольку на практике существенная часть задач, характерных для бизнес-приложений, уже была решена и описана в открытых источниках.

Процесс квалификационного тестирования системы

Цель процесса квалификационного тестирования системы заключается в подтверждении того, что реализация каждого системного требования тестируется на соответствие, а система готова к поставке [1] .

В результате данного процесса должны быть разработаны критерии тестирования, проведено тестирование и задокументированы его результаты.

Различают различные виды тестирования: по степени автоматизации (ручное и автоматическое), по объекту тестирования (функциональное, производительности, безопасности, юзабилити), по степени изолированности (компонентное, интеграционное, системное), по времени проведения (альфа, бета, регрессионное), по степени подготовленности (интуитивное, по документации).

Для управления тестированием используются системы управления тестированием (Test Management System, TMS) реализующие функции планирования тестирования и взаимодействия с артефактами, версионирования планов и артефактов тестирования, оповещения, сбора данных и мониторинга стадии тестирования и качества тестируемого продукта. Среди систем данного класса выделяют: ALM Octane, Test IT, TestRail, Zephyr, Allure, PractiTest, Cerberus Testing, TestLink, Nitrate, Zebrunner, HP Quality Center.

Процесс инсталляции программных средств

Цель процесса инсталляции программных средств заключается в установке программного продукта, удовлетворяющего заданным требованиям, в целевую среду применения [1] .

В рамках данного процесса разрабатывается стратегия инсталляции, критерии проверки соответствия инсталляции требованиям и проводится непосредственная инсталляция. Стратегия инсталляции может быть выражена в виде набора правил или инструкций. В настоящее время инсталляция проводится с использованием подходов «конфигурация как код» (Configuration-as-code, CaC, GitOpts) и инфраструктура как код (IaC), а также инструментов непрерывной интеграции (Continuous Integration). Для реализации данных подходов можно выделить такие инструменты как ArgoCD, Flux, Tekton, Werf, Terraform, SaltStack, Puppet, Chef, Ansible. Среди инструментов непрерывной интеграции можно выделить GitLab CI, Github Actions, Jenkins, TeamCity.

При инсталляции необходимо учитывать следующие, относящиеся к разделению по уровням, факторы:

Использование виртуализации. Виртуализации может подвергаться вся платформа, операционная система, программное обеспечение (виртуальные машины и приложения), память, система хранения данных, сеть и т.д. Операционные системы могут виртуализироваться с помощью гипервизоров, подразделяемых на первый (автономный, тонкий, исполняемый на «голом железе», Type 1, native, bare-metal) и второй (хостовый, монитор виртуальных машин, hosted, Type-2, V) типы [8], а также гибридные. К первому типу могут быть отнесены VMWare ESXi, KVM (может быть отнесен ко второму типу), Xen, Hyper-V, однако, два последних могут быть также отнесены к гибридному типу. Ко второму типу могут быть отнесены Oracle VM VirtualBox, VMWare Workstation.

Использование контейнеризации. Контейнеризация представляет собой виртуализацию на уровне операционной системы, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. Среди основных средств контейнеризации выделяют Docker, Podman, LXC, LXD, runc, crun.

Необходимость оркестрации. В случае, если инсталляция осуществляется с использованием нескольких контейнеров, рационально автоматически размещать, управлять и координировать их взаимодействие, то есть выполнять оркестрацию. Среди средств оркестрации выделяют Kubernetes, Docker Swarm, Docker-compose, Apache Mesos, OpenShift, Nomad. Для указанных средств оркестрации или контейнеров, управляемых ими, могут быть применены средства автоматизации жизненного цикла и построения процессов управления, такие как Conductor, Cloudify, Rancher, Containership, AZK, а также средства управления кластером, такие как k9s, Lens, Kubectl, Marathon.

Указанные факторы непосредственно влияют на соответствие инсталляции требованиям, например, в части надежности и отказоустойчивости системы, а также на сложность процессов реализации и поддержки. В связи с этим добросовестной практикой при инсталляции считается учитывать зависимости между уровнями и определять ответственных за каждый уровень, в том числе аппаратный.

Процесс поддержки приемки программных средств

Цель процесса поддержки приемки программных средств заключается в содействии приобретающей

стороне в обеспечении уверенности в том, что продукт соответствует заданным требованиям [1].

В рамках данного процесса разработчик должен решить следующие задачи:

Скомплектовать, поставить и обеспечить применение системы по назначению. В настоящее время данная задача преимущественно решается с использованием инструментов непрерывной интеграции.

Выполнить поддержку приемочных тестов и ревизий, выполняемых принимающей стороной. Для этого используют системы управления тестированием, стандарты и модели формализации требований, а также системы управления требованиями.

Идентифицировать и обеспечить решение проблем. Для этого используют трекеры задач, а найденные решения документируют с применением wiki-движков, систем управления контентом или систем управления знаниями.

Обеспечить обучение приобретающей стороны. Обучение обеспечивается путем передачи соответствующей документации и проведения обучения, в том числе с использованием систем управления обучением (learning management system, LMS), таких как Moodle, Open edX LMS, Sakai, ILIAS, OpenOLAT.

Процесс функционирования программных средств

Цель процесса функционирования программных средств заключается в применении программного продукта в предназначенной для него среде и обеспечении поддержки заказчиков программного продукта [1] .

Процесс функционирования программных средств выполняется оператором программного средства. На практике оператором часто является заказчик программного обеспечения. Однако, если система предоставляется как услуга (SaaS, software as a service), то оператор, заказчик и пользователи системы могут быть разделены, в том числе являться разными юридическими лицами. При этом каждый уровень среды, в которой функционирует система, также может поставляться в качестве услуги, у которой есть свой оператор (например, система может быть размещена в арендуемой виртуальной инфраструктуре, мощности для которой предоставляет сторонний центр обработки данных) .

В процессе функционирования должны быть определены и оценены условия корректной работы системы, произведены настройки и консультации, в совокупности обеспечивающие функционирование системы в предназначенной для нее среде. Выше были описаны средства доставки, настройки окружения и учета обращений, которые должны обеспечивать нормальное функционирование системы.

В случае инцидентов должна быть обеспечена прослеживаемость причин инцидента, а также должен быть определен ответственный за устранение инцидента и его причин. Для этого используют различные инструменты мониторинга такие как Zabbix, Icinga, Prometheus, Nagios, Netdata, LibreNMS, PagerDuty и др. Указанные инструменты позволяют оперативно реагировать на инциденты, устанавливать их причины и анализировать предпосылки данных причин. На практике особое внимание обращают на мониторинг границ сред и границ организационной ответственности.

Процесс сопровождения программных средств

Цель процесса сопровождения программных средств заключается в обеспечении эффективной по затратам поддержки поставляемого программного продукта [1]. Совокупность характеристик, минимизирующих затраты на устранение ошибок и модификацию согласно ГОСТ Р ИСО/МЭК 14764–2002 определяется как сопровождаемость. Согласно ГОСТ выделяют следующие виды сопровождения: корректирующее (corrective maintenance — корректируются только обнаруженные проблемы), профилактическое (preventive maintenance — корректируются скрытые проблемы в целях их предотвращения), адаптивное (adaptive maintenance — осуществляется модификация для обеспечения работоспособности в меняющейся среде), полное (perfective maintenance — программное обеспечение модифицируется для повышения характеристик и улучшения сопровождаемости).

В каскадной модели сопровождение выделяется в отдельную фазу жизненного цикла, при этом в ГОСТ 34.601–90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания» различают сопровождение в рамках гарантийного и послегарантийного обслуживания. При этом, например, в спиральной модели сопровождение не выделяется как отдельный этап.

Сопровождение программных средств проводится с использованием соответствующего конгломерата инструментальных (вспомогательных) программных средств, методов и технологий программной инженерии.

Таблица 1. Сущности и отношения между ними.

	Стандарт	Процесс	Класс задач	Класс систем	Инструмент
Стандарт	Быть синонимом	Включать в себя Обобщать	Включать в себя Обобщать		
Процесс	Принадлежать Реализовывать	Быть синонимом	Включать в себя	Обобщать	
Класс задач	Принадлежать Реализовывать	Принадлежать	Быть синонимом	Обобщать	
Класс систем		Реализовывать	Реализовывать	Быть синонимом	Включать в себя Обобщать
Инструмент				Принадлежать Реализовывать	Быть синонимом

Среди программных средств можно выделить трекеры задач и системы поддержки (helpdesk) такие как GLPI, Request Tracker, Helpy, osTicket, Faveo, FreeScout, Zammad. Для определения стоимости сопровождения программных средств используются CASE-средства.

Процесс прекращения применения программных средств

Цель процесса прекращения применения программных средств состоит в обеспечении завершения существования системного программного объекта [1] .

В ходе прекращения применения программного средства составляется план, предусматривающий в том числе прекращение поддержки, архивирование или уничтожение данных и компонентов системы, обеспечение доступа к архиву и переход к новому программному продукту.

В случае перехода на другой программный продукт практикуется введение периода, когда производится параллельная работа в прекращаемом и внедряемом программном обеспечении. Параллельное ведение позволяет проверить соответствие данных и расчетов в программном обеспечении (т.е. провести сложные сценарные тесты), уточнить функциональные требования к внедряемому программному обеспечению, а также снизить риски простоя или критических проблем внедряемого программного обеспечения.

После завершения параллельного ведения производится миграция данных из одной системы в другую. Миграция может быть как однократной и непродолжительной по времени, так и многократной (в таком случае можно говорить о первоначальной миграции и дальнейшем информационном обмене) и продолжительной во времени. Для миграции может использоваться как прямое обращение от системы к системе, так и связующее программное обеспечение, ориентированное на обработку сообщений (message-oriented middleware, MOM), например, службы обмена сообщениями (брокеры, message broker, integration broker, interface engine), ETL-системы (Extract, Transform, Load) и сервисные шины данных (enterprise service bus, ESB). Среди служб обмена сообщениями можно выделить Apache Kafka, RabbitMQ, ZeroMQ, IBM MQ, Apache ActiveMQ, KubeMQ. Среди ETL-систем и сервисных шин можно выделить Apache Camel, Apache NiFi, Node-RED, Airbyte, Pentaho, Singer.

Также после прекращении применения производится сохранение знаний о программном обеспечении. Оно необходимо в случаях обращения к архивным данным (например, миграции могут не подлежать первичные данные или миграция могла не проводиться в связи с отсутствием системы-приемника). Для сохранения знаний могут быть использованы wiki-страницы на основе wiki-движков, например, MediaWiki, DokuWiki, Gollum, сочетающие хранение wiki-страниц с управлением версиями через git, или специализированная система управления знаниями (Knowledge Management Systems, KMS), такая как Atlassian Confluence, Notion, AFFiNE, Obsidian, Helpy или Documize.

После прекращения применения один или несколько экземпляров программного обеспечения могут быть сохранены в работоспособном состоянии для обеспечения доступа к архивным данным. В некоторых случаях для этого используются эмуляторы или среды виртуализации, такие как DOSBox, QEMU, Bochs, Proxmox VE, OpenVZ, Xen, oVirt, а также описанные в разделе «Процесс инсталляции программных средств» средства.

Заключение

В статье рассмотрены процессы группы «Технические процессы» ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств» и исследованы концептуальные отношения —

квалитативные и квантитативные в разрезе следующих понятийных сфер: абстрактное-конкретное, принадлежность, форма и содержание, процессуальность, тождество и противоречие. Дан краткий обзор инструментов, а также некоторых методов и практических приемов применяемых для решения задач, относящихся к категории процессов проекта.

Исследование источников проведено с применением подхода «слоеный пирог обучения онтологий» [14]: в текстах были выделены сопоставляемые инструментам термины и отношения между ними, далее выделены концепты, сопоставленные классам систем и задач. Выделение проводилось по эквивалентным аксиомам процессам стандарта ГОСТ Р ИСО/МЭК 12207–2010.

Основные выявленные сущности и отношения, которые могут быть использованы при построении семантической модели приведены в Таблице 1.

Процессы являются частью стандартов (принадлежность, отношение агрегации), вместе с тем, процессы разных стандартов могут коррелировать, или быть синонимами (тождество и противопоставление) .

Стандарты включают (принадлежность, отношение агрегации) в себя процессы и классы задач. Процессы могут включать классы задач и обобщать (абстрактное-конкретное, отношение иерархии «Род-Вид») классы систем. Классы задач обобщают классы систем. Классы систем реализуют процессы и классы задач, включают в себя и обобщают инструменты. Все сущности могут являться синонимами друг друга.

Таким образом, анализ процессов, проведенный в двух частях статьи [9] позволил выявить ряд сущностей и основных отношений между ними и провести их первичную систематизацию.

Статья является второй частью работы по анализу процессов жизненного цикла программного обеспечения. В следующей части предполагается провести обзор процессов категории «Процессы реализации программных средств».

ЛИТЕРАТУРА

1. ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»
2. 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.
3. Травасолли Д. Управление требованиями. Десять шагов к совершенству. [Электронный ресурс] URL: https://www.swd.ru/files/pdf/IBM_uspeh_edit1.pdf (дата обращения: 28.11.2022)
4. Приемы управления требованиями к ПО. [Электронный ресурс] URL: https://analytics.infozone.pro/requirements-analysis/requirements-management-methods/ (дата обращения: 28.11.2022)
5. А.Ф. Галимянов. Архитектура информационных систем / А.Ф. Галимянов Ф.А. Галимянов — Казань, Казан. ун-т, 2019–117 с.
6. ГОСТ Р 57100–2016/ISO/IEC/IEEE 42010:2011 «Системная и программная инженерия. Описание архитектуры».
7. Р. Мартин Чистая архитектура. Искусство разработки программного обеспечения / Пер. с англ.: А. Киселев.— СПб.: Питер, 2022.— 352 с.— ISBN 978–5–4461–0772–8.
8. Gerald J. Popek, Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures [Электронный ресурс] URL: https://dl.acm.org/doi/pdf/10.1145/361011.361073 (дата обращения: 11.01.2023)
9. Семантическое моделирование: обзор процессов, инструментов, методов и знаний предметной области разработки программного обеспечения (Часть 1) / А.Н. Тимофеев, И.С. Евдокимова, Н.Б. Хаптахаева, А.А. Сенотрусова // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки.— 2022.— № 12.— С. 92–97.
10. ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering
11. ГОСТ Р ИСО/МЭК 25051–2017 «Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Требования к качеству готового к использованию программного продукта (RUSP) и инструкции по тестированию»
12. ГОСТ Р ИСО/МЭК 25010–2015 «Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов»
13. ИСО 31000:2018 «Менеджмент риска. Принципы и руководство» (ISO 31000:2018 «Risk management — Guidelines», IDT) .
14. Asim M.N., Wasim M., Khan M.U.G., Mahmood W., Abbasi H.M.A survey of ontology learning techniques and applications. Database, 2018. DOI: 10.1093/database/bay101.

© Тимофеев Александр Николаевич (tan@sibdigital.net), Евдокимова Инга Сергеевна (evdinga@gmail.com),
Хаптахаева Наталья Баясхалановна (khapnb@gmail.com).
Журнал «Современная наука: актуальные проблемы теории и практики»