Научная статья
УДК 004.82
https://doi.org/10.24143/2072-9502-2023-2-66-74
EDN IFNMOY

Онтология жизненного цикла разработки программного обеспечения

Татьяна Эриковна Шульга ✉, ***Дмитрий Эдуардович Храмов***

Саратовский государственный технический университет имени Гагарина Ю. А.,
Саратов, Россия, taiss@yandex.ru ✉

Аннотация. Рассматривается проблема представления знаний о моделях жизненного цикла программного обеспечения (ПО), необходимость решения которой обусловлена стремительным развитием методологий разработки ПО, отсутствием формальной легко расширяемой модели знаний в этой предметной области и тем, что выбор модели жизненного цикла и соответствующей ей методологии разработки оказывает значительное влияние на успешность программных проектов. Проведен системный анализ основных типов методологий разработки ПО, моделей жизненного цикла и их фаз. Приведены результаты исследования области представления моделей жизненного цикла ПО в виде онтологий. Разработана онтология «Software development life cycle (SDLC)», которая предназначена для представления знаний о различных моделях жизненного цикла ПО, фазах (стадиях) жизненного цикла, присущих различным моделям, и возможности описания повторяемости фаз. Онтология позволяет описывать модели как в рамках прогностических методологий разработки (водопадная, инкрементная), так и в рамках гибких методологий разработки (Scrum, Kanban). Описаны классы, свойства и аксиомы онтологии, на основе которых возможно осуществление формального логического вывода. Онтология SDLC разработана на основе форматов семантического веба (на языке OWL), опубликована в открытом доступе и представляет собой развивающийся, легко расширяемый проект. Это позволит использовать ее любым специалистам в области разработки ПО в практических или исследовательских целях. Также представлена идея программной оболочки, использующей представленную онтологию, которая позволит по заданным параметрам выбрать наиболее подходящую методологию для проекта, что упростит процесс разработки, позволит избежать ряда ошибок и сократит время на разработку.

Ключевые слова: программное обеспечение, методология, модель, разработка программного обеспечения, жизненный цикл разработки ПО, модель жизненного цикла ПО, Agile, Scrum, Kanban

Для цитирования: *Шульга Т. Э., Храмов Д. Э.* Онтология жизненного цикла разработки программного обеспечения // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика 2023. № 2. С. 66–74. https://doi.org/10.24143/2072-9502-2023-2-66-74 EDN IFNMOY.

Original article

Life cycle ontology of software engineering

Tatiana E. Shulga ✉, ***Dmitry E. Khramov***

Yuri Gagarin State Technical University of Saratov,
Saratov, Russia, taiss@yandex.ru ✉

Abstract. The article highlights the problem of presenting knowledge on the models of software life cycle, the importance of which can be explained by the rapid progress of software engineering methods, by the absence of a formally easily extensible knowledge model in this subject area, and by the fact that cycle time selection models and the proposed development methodology have a significant impact on the success of software projects. System analysis of the main types of software development methodologies, life cycle models and their phases has been carried out. The results of studying the representation of software life cycle models in the form of ontologies are presented. The ontology “Software development life cycle” (SDLC) has been developed. It is designed to represent knowledge about various models of the software life cycle, phases (stages) of the life cycle inherent in different models, and the possibility of describing the recurrence of phases. The ontology allows describing models both within predictive development methodologies (waterfall, incremental) and within agile development methodologies (Scrum, Kanban). Classes, properties and axioms of the ontology are described, on the basis of which it is possible to produce a formal logical inference. The SDLC ontology is developed on top of the Semantic Web formats (in OWL language), published in the public domain and presents a developing, easily extensible project. This can probably be used in the field of software

development for practical or research purposes. There is also introduced the idea of a software shell that uses the presented ontology, which will allow, according to the given parameters, to choose the most appropriate methodology for the project, which will simplify the development process, avoid errors and reduce development time.

Keywords: software, methodology, model, software engineering, software engineering life cycle, software life cycle model, Agile, Scrum, Kanban

For citation: Shulga T. E., Khramov D. E. Life cycle ontology of software engineering. *Vestnik of Astrakhan State Technical University. Series: Management, computer science and informatics*. 2023;2:66-74. (In Russ.). https://doi.org/10.24143/2073-5529-2023-2-66-74. EDN IFNMOY.

Введение

Современное разнообразие методологий, методов и технологий разработки программного обеспечения (ПО) объясняется во многом стремлением увеличить процент успешных программных проектов. Под успешным программным проектом понимают проект, выполненный в разумные прогнозируемые сроки, в рамках установленного бюджета и обеспечивающий удовлетворенность заказчика вне зависимости от первоначального масштаба проекта. Согласно статистике [1], выбор методологии разработки ПО оказывает значительное влияние на успешность программных проектов. При этом центральным понятием в методологиях разработки является понятие модели жизненного цикла ПО (software development life cycle, SDLC). Несмотря на существование различных классификаций моделей SDLC и отдельных работ, сравнивающих разные методологии разработки и модели (например, [2, 3]), в научной литературе отсутствует системный подход к представлению знаний в этой предметной области.

Для решения проблемы авторами проведен системный анализ существующих методологий и моделей разработки ПО и как результат этого анализа выполнена формализация знаний в данной предметной области в виде онтологии. Применение популярного онтологического подхода (например, [4–6]) обосновано, прежде всего, гибкостью онтологического моделирования, возможностью легкого расширения модели при появлении новых методологий разработки ПО и возможностями «простого» представления модели в открытых форматах. Эти преимущества позволят использовать модель различными исследователями, в том числе для решения практических задач: по заданным параметрам выбрать наиболее подходящую методологию для проекта, анализировать время продолжительности разработки ПО. В частности, авторами осуществляется разработка программной оболочки, позволяющей на основе представленной онтологии автоматизировать процесс выбора методологии для конкретного программного проекта.

Краткий обзор предметной области

Анализ большого числа научных источников и практических руководств по разработке ПО показал, что в исследуемой области существует терминологическая путаница, а именно термины «методология разработки ПО» и «модель разработки ПО» часто используются как синонимы. Если обращаться к общепринятому пониманию этих терминов (например, в Большой советской энциклопедии, словаре русского языка Ожегова), то относительно процесса разработки ПО их можно трактовать следующим образом: методология – совокупность методов, применяемых при разработке ПО, модель – формальное описание процесса разработки ПО. Согласно этому именно модели содержатся в основе определенной методологии. Отметим, что, если на «заре развития» программной инженерии в литературе чаще использовался термин «модель» (например, водопадная модель, инкрементная модель), то в последние годы он фактически заменен на термин «методология» (например, XP-методология, методология Scrum). Тем не менее, будем использовать термин «модель», т. к. нас интересуют не конкретные методы разработки, а именно формальное описание процесса разработки.

Длительное время при разработке ПО использовались прогностические методологии (one-driven), т. е. методологии, в основе которых содержится идея о том, что требования к ПО должны быть собраны на начальном этапе разработки и не могут изменяться в ее процессе. Наиболее известной моделью в рамках прогностической методологии является водопадная (каскадная) модель [7], которая представляет процесс разработки в виде последовательности фаз (этапов), каждый из которых выполняется только один раз без возможности возврата к предыдущим. Другой популярной моделью является инкрементная модель [8, 9], основная идея которой – разработка ПО в виде итераций: в первую итерацию реализуется основной функционал ПО, далее с каждым новым инкрементом продукт совершенствуется, приближаясь к описанному в техническом задании.

В настоящее время популярной становится методология гибкой разработки (Agile) [10]. Наиболее известными и распространенными моделями являются Scrum, Kanban [11]. По сути это итеративный подход к разработке, схожий с инкрементной моделью. В результате выпускается не полноценное ПО, а происходит постоянная поставка некоторых инкрементов, конечно, при этом после выпуска инкремента допускается изменение требований к ПО.

В научной литературе (например, в [9]) и практической деятельности при описании разработки

ПО принято использовать термины «фаза» и «повторяемость фазы». Фаза – это определенный период (этап) жизненного цикла разработки ПО, применяемый в модели. В различных современных моделях могут быть специфичные фазы, такие как фаза sprint для Scrum модели, являющаяся, в сущности, фрагментом инкрементной модели. Повторяемость фазы обозначает возможность повторить в модели определенную фазу. Так, например, в водопадной модели отсутствует повторяемость фаз. Отметим, что синонимом термина «фаза» является термин «стадия» [12].

Основными документами, описывающими разработку ПО на уровне стандартов, являются стандарт ISO на процессы жизненного цикла и соответствующий российский ГОСТ [12]. Важно отметить, что трудно установить однозначное соответствие между понятиями «фаза» и «процесс». Согласно [12], «…процесс – совокупность взаимосвязанных или взаимодействующих видов деятельности, преобразующих входы в выходы» [12, с. 4], этапы разработки находятся в раздельных группах процессов согласно таблице п. 5.2.1 «Категории процессов жизненного цикла». Так, например, основные фазы жизненного цикла ПО могут быть соотнесены с категориями технических процессов, однако планирование находится в процессах проекта. В ГОСТе описаны 43 процесса, а основных фаз, используемых в современных моделях, меньше. В целом, можно утверждать, что одна фаза может включать несколько процессов, а один процесс, например документирование, может реализовываться на разных фазах, причем в ГОСТе не упоминаются процессы, соответствующие специальным фазам, характерным для Agile-моделей.

Анализ существующих онтологий

В сфере разработки ПО использование онтологического подхода достаточно популярно. Например, следует отметить работу [6], в которой описана система поддержки принятия решений на уровне инженерии требований на основе онтологического подхода. В работе представлена онтология, которая «…хранит информацию о требованиях, артефактах требований, источниках требований, включая заинтересованных лиц, и участниках команды, добавляющих требования в форме знаний об экземплярах классов» [6, с. 49].

В работе [13] представлен обзор исследований, связанных с областью разработки ПО, из нее понятно, что на 2014 г. имелось множество упоминаний, связанных с онтологиями для управления проектами. Однако эти работы не связаны с представлением методологий или моделей в виде общей онтологии по SDLC и предназначены для представления знаний о методах управления проектами.

Отдельно следует указать ряд других работ [14–16], связанных с определенными моделями жизненного цикла ПО или целями в рамках программной инженерии, однако в них представлены наработки, связанные исключительно с конкретной моделью жизненного цикла ПО, и они являются частным случаем онтологии по SDLC, например для модели Feature Driven Development [14].

Проведенный анализ показал, что первой и фактически единственной онтологией, которая посвящена именно SDLC, является онтология ODYSSEY [17]. При этом ODYSSEY имеет несколько основных классов: Developer и SDLC, который включает подкласс Модель (Model) и Фаза (Phase), т. е. этап жизненного цикла в рамках конкретной модели. Наличие класса SDLC позволяет рассматривать онтологию как каркас для дальнейших разработок. Кроме этого, имеется 9 объектных свойств, которые показывают как отношения в рамках класса SDLC, так и связь между классами Developer и Model. В онтологии имеется только одно свойство данных, предназначенное для представления названия методологии. Недостаток этой онтологии заключается в том, что она, по словам авторов, промежуточная, и в ней представлена реализация лишь двух вариантов водопадной модели. Таким образом, в ней не учитываются различные нюансы, которые необходимы для других моделей разработки, а особенно моделей гибкой разработки (Agile), что крайне важно для современных IT-компаний. Кроме этого, непонятно назначение класса Developer, который описывает интеллектуальных агентов: людей и искусственные системы, если декларируется, что основная цель онтологии – описание концепции SDLC. Также остается неясным назначение класса SDLC, т. к. ни модель, ни фаза не являются SDLC, и, с нашей точки зрения, это является логической ошибкой. Кроме этого, необходимо отметить, что в работе имеется проблема с описанием класса тестирования: в онтологии установлена эквивалентность между обобщенным классом тестирования (Testing) и классом интеграционного и системного тестирования (Integration_and_System_Testing), а класс имплементации совместно с модульным тестированием (Implemetation_and_Unit_Testing), в свою очередь, эквивалентен классу кодирования (Coding). С нашей точки зрения, это является ошибкой, т. к. в некоторых моделях предполагается исключительно последовательное прохождение фаз в определенном порядке.

Кроме ODYSSEY имеется онтология [18], которая посвящена тестированию. В ней представлены различные аспекты этого этапа: от документации до соответствующих возможных фаз тестирования. В отличие от онтологии ODYSSEY, фазы тестирования представлены отдельными классами. В дополнение следует отметить, что кроме классов

тестирования в онтологии представлен отдельный класс для SDLC, однако в самой работе он не рассмотрен и неизвестно его конечное назначение.

Таким образом, несмотря на разнообразие существующих онтологий, в научных источниках не найдено онтологии, которая описывает фазы жизненного цикла ПО для различных моделей и может быть использована для выбора модели разработки, на основе заданных характеристик конкретного программного проекта.

Онтология SDLC

Учитывая описанное, принято решение о разработке онтологии «Software development life cycle (SDLC)», позволяющей представить знания о различных моделях жизненного цикла ПО. Разработка осуществлялась на основе онтологии ODYSSEY в соответствии со следующими соображениями:

1. Структура классов онтологии SDLC разрабатывается на основе структуры классов онтологии ODYSSEY, т. к. последняя содержит некоторые из общепринятых фаз и типов моделей.

2. Необходимо добавить соответствующие классы для представления знаний о различных моделях (в ODYSSEY реализована только водопадная модель).

3. Онтология разрабатывается с учетом дальнейшей реализации программной оболочки над ней (упомянутой выше).

4. Необходимо добавить свойства онтологии, которые используются в оболочке.

Разработка онтологии осуществлялась на языке онтологического моделирования OWL в свободно распространяемом редакторе онтологий Protége.

В качестве основы взята структура классов онтологии ODYSSEY с сопутствующими изменениями в классах для фаз. Например, решено отказаться от класса Developer и родительского класса SDLC. Кроме этого, добавлены специальные фазы для модели Scrum и Kanban. Изменения также коснулись фазы тестирования.

Таким образом, классовая структура онтологии, представленной на рис. 1, состоит из родительского класса Model, который содержит сведения о моделях гибкой разработки (дочерний класс Agile model) и о прогностических моделях (дочерний класс Plan Driven Model); родительского класса Phase, который включает дочерние классы фаз, используемые в различных моделях.

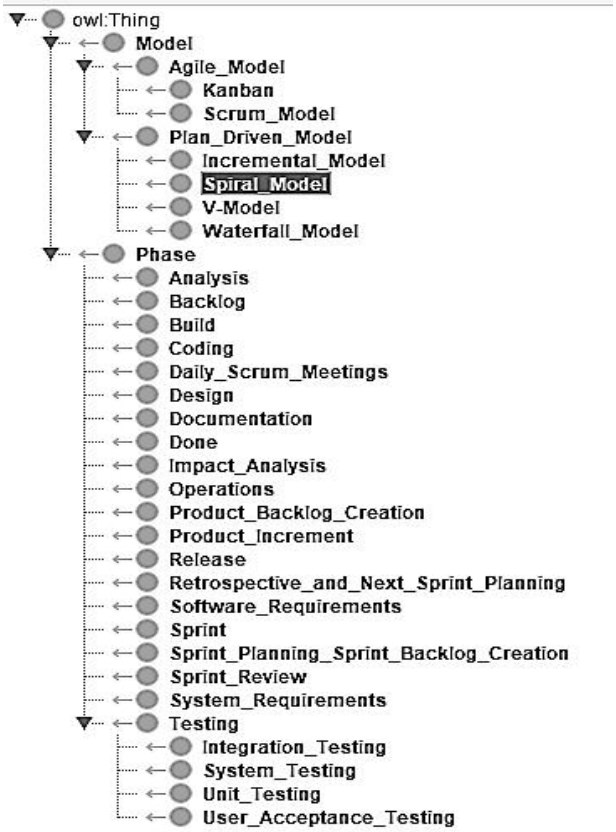

Рис. 1. Классы онтологии

Fig. 1. Classes of ontology

В таблице приведен пример описания классов онтологии, в частности некоторых подклассов класса Phase.

Пример описания классов онтологии

Example of ontology classes description

Класс	Назначение класса
Analysis (анализ)	Фаза для анализа системных требований (специальный случай процесса анализа требований) [12]
Coding (кодирование)	Соответствует процессу конструирования. Процесс конструирования программных средств является процессом более низкого уровня, чем процесс реализации программных средств [12]
Design (проектирование)	Фаза проектирования архитектуры системы (специальный случай процесса проектирования архитектуры) [12]
Sprint (спринт)	Фаза, на которой берется предварительно выделенный объем работы из журнала задач. Спринт заканчивается демонстрацией новых функций. Эта фаза включает в качестве подфаз фрагмент инкрементной модели [19–21]
Sprint Planning, Backlog Creation (планирование спринта, создание журнала задач)	Фаза, на которой определяются сроки спринта, а также цель и список задач [21]
Testing (тестирование)	Тестирование ПО состоит из динамической проверки того, что программа обеспечивает ожидаемое поведение в конечном наборе тестовых случаев. Соответствует процессу квалификационного тестирования программных средств [12, 22]
Integration Testing (интеграционное тестирование)	Интеграционное тестирование – это процесс проверки взаимодействия между программными компонентами. Данный класс является подклассом класса Testing [22]
System Testing (системное тестирование)	Системное тестирование связано с тестированием поведения всей системы. Системное тестирование обычно считается подходящим для оценки нефункциональных системных требований, таких как безопасность, скорость, точность и надежность. Данный класс является подклассом класса Testing [22]
Unit Testing (модульное тестирование)	Модульное тестирование проверяет функционирование отдельных программных элементов, которые можно тестировать по отдельности. В зависимости от контекста это могут быть отдельные подпрограммы или более крупный компонент, состоящий из взаимосвязанных единиц. Данный класс является подклассом класса Testing [22]

Если сравнивать классовую иерархию предлагаемой онтологии и исходной, то в предложенной нами онтологии виды тестирования представлены в виде подкласса Testing. Это решение обусловлено тем, что в некоторых моделях осуществляется последовательное выполнение фаз тестирования без возврата к предыдущим этапам. В исходной онтологии ODYSSEY выполнена разработка исключительно под различные версии водопадной модели.

Кроме этого, в онтологии SDLC определены объектные свойства и свойства данных. Некоторые из них заимствованы для разработки, однако, учитывая, что в ODYSSEY не имеется отдельных свойств для Agile моделей и не хватает возможности для описания других прогностических моделей, здесь также имеется ряд изменений и дополнений.

Одними из главных ключевых элементов онтологии, которые необходимы для дальнейшей разработки оболочки по выбору подходящей модели разработки, являются свойства данных. В представленной нами модели имеется несколько логических свойств типа xsd:boolean, которые отвечают за следующие параметры:

– isRepeatable – повторяемость, может повторяться определенная фаза;

– canBackToPhase – возврат к фазе, имеется возможность в модели вернуться на шаг назад;

– hasFinalModule – имеет конечный модуль, на текущем цикле итерации разработчики получают конечный модуль или определенный каркас.

Также определен ряд объектных свойств, которые указывают на взаимодействие между классами, такие как:

– dividedBy – показывает, на какие подфазы может разбиваться текущая фаза; свойство актуально для моделей гибкой разработки;

– hasPhase – позволяет указать фазы, которые имеются в модели;

– hasToMeet – свойство показывает фазу, которая выполняется следующей и для которой необходимо начать/выполнить текущую;

– isDependentOf – свойство, которое задает фазу, которую необходимо начать/выполнить до того, как перейти к текущей.

Последние два свойства являются обратными, т. е. если указали, что текущая фаза необходима для выполнения следующей, но для нее не указали hasToMeet, то машина логического вывода (ризенер) укажет это свойство (рис. 2) и наоборот.

Рис. 2. Пример работы ризенера. Связь фаз

Fig. 2. Example of a reasoner's operation. Phase connection

Указанные свойства позволяют упростить дальнейший выбор, используя их значения, и однозначно «отсечь» ненужные модели при работе с оболочкой. Например, при указании, что мы предполагаем возврат к фазе, из списка предполагаемых моделей исключаются водопадная, V-model и инкрементная.

Что касается объектных свойств, то в нашем случае их количество сокращено, т. к. в ODYSSEY имелся ряд свойств, которые связаны с классом Developer.

В настоящий момент времени в онтологии реализовано две модели гибкой разработки – Scrum, Kanban, и четыре прогностических модели: водопадная, спиральная, инкрементная и V-model. Пример описания спиральной модели представлен на рис. 3.

Фактическое представление спиральной модели позволяет увидеть ее фазы, а также два свойства данных, рассмотренных выше: во время работы разработчики могут вернуться к прошедшей фазе; по результатам завершения итерации разработчики получают не конечный модуль, а прототип.

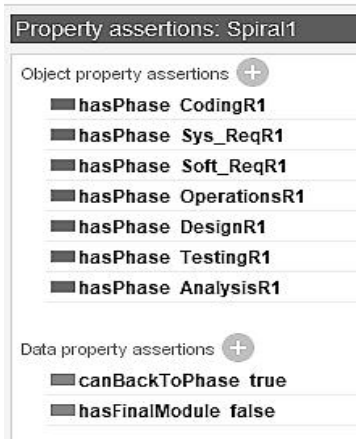

Рис. 3. Описание спиральной модели

Fig. 3. Description of the spiral model

Пример работы ризенера на этой модели представлен на рис. 4.

Рис. 4. Пример работы ризенера. Схожие классы

Fig. 4. Example of a reasoner's operation. Similar classes

В результате логического вывода ризенер соотносит инкрементную модель с остальными прогностическими. Эти модели в общем плане описания похожи, т. к. состоят из одинаковых фаз. Основными их отличиями являются возможность возврата к фазе, ее повтор и результат завершения итерации.

Еще один пример работы ризенера представлен на рис. 5.

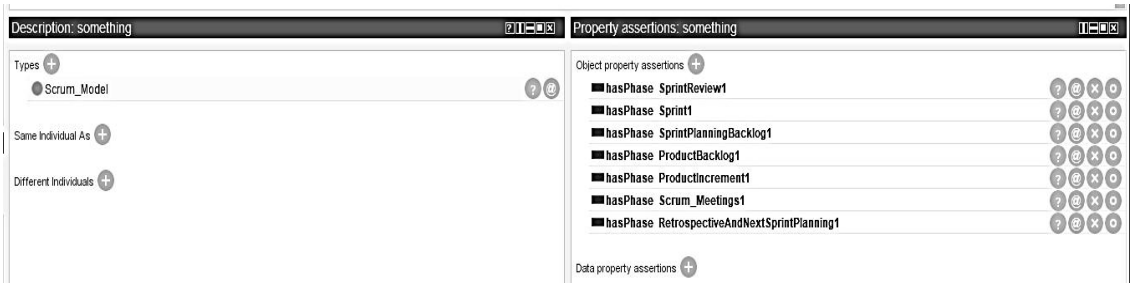

Рис. 5. Пример работы ризенера. Определение класса

Fig. 5. Example of a reasoner's operation. Class definition

Создана сущность без принадлежности к классу, и машина логического вывода определила ее к классу scrum моделей.

Над предложенной онтологией планируется реализовать программную оболочку на языке программирования Python. Пользовательский интерфейс будет доступен на двух языках: русском и английском. Для выбора подходящей модели разработки предполагается, что пользователь вводит ряд параметров, на основе которых происходит предварительная фильтрация моделей. К основным входным переменным можно отнести полноту исходных данных, изменяемость требований, возможность повтора или возврат к фазам.

Заключение

Разработана онтология SDLC, которая предназначена для представления знаний о различных моделях жизненного цикла ПО, в том числе моделях гибкой разработки Scrum и Kanban. Онтология позволяет накапливать в форме экземпляров классов информацию о моделях жизненного цикла и его фазах, показывая их связь.

На основе онтологии SDLC авторами планируется разработать программную оболочку, которая позволит по заданным параметрам выбрать наиболее подходящую модель жизненного цикла для проекта. Это, в свою очередь, должно упростить процесс разработки, а также позволит избежать ряда ошибок и сократит время на разработку. Кроме этого, предполагаем, что она будет являться частью системы для анализа продолжительности жизненного цикла ПО. Дополнение предложенной онтологии большим числом моделей позволит повысить ее эффективность.

Онтология SDLC представлена в отрытых форматах семантического веба (на языке OWL) и опубликована в открытом доступе [23]. Это позволит использовать ее любым специалистам в области разработки ПО для разработки приложений в различных практических или исследовательских целях. Авторы открыты к сотрудничеству по развитию онтологии и будут благодарны экспертам за соответствующие замечания и предложения.

Список источников

1. CHAOSReport 2015. URL: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (дата обращения: 03.10.2022).

2. Stoica M., Mircea M., Ghiliv-Micu B. Software Development: Agile vs. Traditional // Informatica Economică. 2013. V. 17. N. 4. P. 64–76. DOI: 10.12948/issn14531305/17.4.2013.06.

3. Elshandidy H., Mazen Sh. Agile and Traditional Requirements Engineering: A Survey // International Journal of Scientific & Engineering Research. 2013. V. 4. Iss. 9. P. 473–482.

4. Сытник А. А., Шульга Т. Э., Данилов Н. А. Онтология предметной области «Удобство использования программного обеспечения» // Тр. Ин-та систем. программирования РАН. 2018. Т. 30. № 2. С. 195–214.

5. Шульга Т. Э. Никулина Ю. А. Онтологическая модель предметной области «Системы противопожарной безопасности» // Изв. СПбГТИ (ТУ). 2019. № 51 (77). С. 109–114.

6. Муртазина М. Ш. Система поддержки принятия решений при гибком подходе к инженерии требований на основе owl-онтологии // Вестн. Астрахан. гос. техн. ун-та. Сер.: Управление, вычислительная техника и информатика. 2018. № 4. С. 43–55.

7. Royce W. W. Managing the Development of Large Software Systems // Proceedings of IEEE WESCON. 1970. V. 26. P. 328–388.

8. Basili V., Turner J. Iterative enhancement: A practical technique for software development // IEEE Trans. Softw. Eng. 1975. V. 1. N. 4. P. 390–396.

9. Basili V., Larman C. Iterative and incremental development: A brief history // IEEE Comput. Soc. 2003. V. 36. N. 6. P. 47–56.

10. Manifesto for Agile Software Development. URL: http://agilemanifesto.org/ (дата обращения: 23.11.2022).

11. Yadav N. S., Goar V., Kuri M. Agile Methodology - a Perfect Sdlc Model with Some Improvements // Journal

of Critical Reviews. 2020. V. 7. P. 2511–2514. DOI: 10.31838/jcr.07.19.306.

12. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. М.: Стандартинформ, 2011. 99 с.

13. Fitsilis P., Gerogiannis V., Anthopoulos L. Ontologies for Software Project Management: A Review // Journal of Software Engineering and Applications. 2014. V. 7. P. 1096–1110. DOI: 10.4236/jsea.2014.713097.

14. Siddiqui F., Alam A. Ontology Based Feature Driven Development Life Cycle // International Journal of Computer Science. 2013. Iss. 9. URL: https://arxiv.org/ftp/arxiv/papers/1307/1307.4174.pdf (дата обращения: 19.11.2022).

15. Lin Y., Hilaire V., Gaud N., Koukam A. Scrum Conceptualization Using K-CRIO Ontology // International Symposium on Data-Driven Process Discovery and Analysis SIMPDA 2011, Lecture Notes in Business Information Processing book series LNBIP 116. 2012. P. 189–211. DOI: 10.1007/978-3-642-34044-4_11.

16. Zada I., Shahzad S., Ali Sh., Mehmood R. M. OntoSuSD: Software engineering approaches integration ontology for sustainable software development // Software: Practice and Experience. 2022. P. 1–35. DOI: 10.1002/spe.3149.

17. Olszewska J., Allison I. ODYSSEY: Software Development Life Cycle Ontology // Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management – KEOD. 2018. P. 303–311. DOI: 10.5220/0006957703030311.

18. Olszewska J. AI-T: Software Testing Ontology for AI-based Systems // Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2020). 2020. V. 2. P. 291–298. DOI: 10.5220/0010147902910298.

19. Schwaber K. Agile Project Management with Scrum // Developer Best Practices. Microsoft Press, 2004. P. 192.

20. Beedle M., Devos M., Sharon Y., Schwaber K., Sutherland J. SCRUM: an extension pattern language for hyperproductive software development // Pattern Languages of Program Design. 1999. V. 4. P. 637–651.

21. Sharma Sh., Sarkar D., Gupta D. Agile Processes and Methodologies: A Conceptual Study // International Journal on Computer Science and Engineering. 2021. V. 4. N. 05. P. 892–898.

22. Bourque P., Fairley R. D. SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge // IEEE. 2014. P. 335.

23. SDLC Ontology. URL: https://github.com/Dmitry2571/SDLC (дата обращения: 08.12.2022).

References

1. *CHAOSReport 2015.* Available at: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (accessed: 03.10.2022).

2. Stoica M., Mircea M., Ghiliv-Micu B. Software Development: Agile vs. Traditional. *Informatica Economică*, 2013, vol. 17, no. 4, pp. 64-76. DOI: 10.12948/issn14531305/17.4.2013.06.

3. Elshandidy H., Mazen Sh. Agile and Traditional Requirements Engineering: A Survey. *International Journal of Scientific & Engineering Research*, 2013, vol. 4, iss. 9, pp. 473-482.

4. Sytnik A. A., Shul'ga T. E., Danilov N. A. Ontologiia predmetnoi oblasti «Udobstvo ispol'zovaniia programmnogo obespecheniia» [Ontology of subject area “Ease of using software”]. *Trudy instituta sistemnogo programmirovaniia RAN*, 2018, vol. 30, no. 2, pp. 195-214.

5. Shul'ga T. E. Nikulina Iu. A. Ontologicheskaia model' predmetnoi oblasti «Sistemy protivopozharnoi bezopasnosti» [Ontological model of subject area “Fire safety systems”]. *Izvestiia SPbGTI (TU)*, 2019, no. 51 (77), pp. 109-114.

6. Murtazina M. Sh. Sistema podderzhki priniatiia reshenii pri gibkom podkhode k inzhenerii trebovanii na osnove owl-ontologii [Decision support system for flexible approach to requirements engineering based on owl-ontology]. *Vestnik Astrakhanskogo gosudarstvennogo tekhnicheskogo universiteta. Seriia: Upravlenie, vychislitel'naia tekhnika i informatika*, 2018, no. 4, pp. 43-55.

7. Royce W. W. Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, 1970, vol. 26, pp. 328-388.

8. Basili V., Turner J. Iterative enhancement: A practical technique for software development. *IEEE Trans. Softw. Eng*., 1975, vol. 1, no. 4, pp. 390-396.

9. Basili V., Larman C. Iterative and incremental development: A brief history. *IEEE Comput. Soc*., 2003, vol. 36, no. 6, pp. 47-56.

10. *Manifesto for Agile Software Development*. Available at: http://agilemanifesto.org/ (accessed: 23.11.2022).

11. Yadav N. S., Goar V., Kuri M. Agile Methodology - a Perfect Sdlc Model with Some Improvements. *Journal of Critical Reviews*, 2020, vol. 7, pp. 2511-2514. DOI: 10.31838/jcr.07.19.306.

12. GOST R ISO/MEK 12207-2010. *Informatsionnaia tekhnologiia. Sistemnaia i programmnaia inzheneriia. Protsessy zhiznennogo tsikla programmnykh sredstv* [GOST R ISO/IEC 12207-2010. Information technology. System and software engineering. Software Life Cycle Processes]. Moscow, Standartinform Publ., 2011. 99 p.

13. Fitsilis P., Gerogiannis V., Anthopoulos L. Ontologies for Software Project Management: A Review. *Journal of Software Engineering and Applications*, 2014, vol. 7, pp. 1096-1110. DOI: 10.4236/jsea.2014.713097.

14. Siddiqui F., Alam A. Ontology Based Feature Driven Development Life Cycle. *International Journal of Computer Science*, 2013, iss. 9. Available at: https://arxiv.org/ftp/arxiv/papers/1307/1307.4174.pdf (accessed: 19.11.2022).

15. Lin Y., Hilaire V., Gaud N., Koukam A. Scrum Conceptualization Using K-CRIO Ontology. *International Symposium on Data-Driven Process Discovery and Analysis SIMPDA 2011, Lecture Notes in Business Information Processing book series LNBIP 116*, 2012, pp. 189-211. DOI: 10.1007/978-3-642-34044-4_11.

16. Zada I., Shahzad S., Ali Sh., Mehmood R. M. OntoSuSD: Software engineering approaches integration ontology for sustainable software development. *Software: Practice and Experience*, 2022, pp. 1-35. DOI: 10.1002/spe.3149.

17. Olszewska J., Allison I. ODYSSEY: Software Development Life Cycle Ontology. *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management – KEOD*, 2018. Pp 303-311. DOI: 10.5220/0006957703030311.

18. Olszewska J. AI-T: Software Testing Ontology for AI-based Systems. *Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2020)*, 2020. Vol. 2. Pp. 291-298. DOI: 10.5220/0010147902910298.

19. Schwaber K. Agile Project Management with Scrum. *Developer Best Practices*. Microsoft Press, 2004. P. 192.

20. Beedle M., Devos M., Sharon Y., Schwaber K., Sutherland J. SCRUM: an extension pattern language for hyperproductive software development. *Pattern Languages of Program Design*, 1999, vol. 4, pp. 637-651.

21. Sharma Sh., Sarkar D., Gupta D. Agile Processes and Methodologies: A Conceptual Study. *International Journal on Computer Science and Engineering*, 2021, vol. 4, no. 05, pp. 892-898.

22. Bourque P., Fairley R. D. SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge. *IEEE*, 2014, p. 335.

23. *SDLC Ontology*. Available at: https://github.com/Dmitry2571/SDLC (accessed: 08.12.2022).

Статья поступила в редакцию 17.01.2023; одобрена после рецензирования 03.03.2023; принята к публикации 18.04.2023
The article is submitted 17.01.2023; approved after reviewing 03.03.2023; accepted for publication 18.04.2023

Информация об авторах / Information about the authors

Татьяна Эриковна Шульга – доктор физико-математических наук, профессор; профессор кафедры информационно-коммуникационных систем и программной инженерии; Саратовский государственный технический университет имени Гагарина Ю. А.; taiss@yandex.ru

Tatiana E. Shulga – Doctor of Physico-Mathematical Sciences, Professor; Professor of the Department of Information and Communication Systems and Software Engineering; Yuri Gagarin State Technical University of Saratov; taiss@yandex.ru

Дмитрий Эдуардович Храмов – аспирант кафедры прикладных информационный технологий; Саратовский государственный технический университет имени Гагарина Ю. А.; dmitriy-hramov@list.ru

Dmitrii E. Khramov – Postgraduate Student of the Department of Applied Information Technologies; Yuri Gagarin State Technical University of Saratov; dmitriy-hramov@list.ru