УДК 372.862 DOI 10.21777/2500-2112-2025-2-45-55

ОПЫТ МОДЕРНИЗАЦИИ МЕТОДОЛОГИИ ICONIX ДЛЯ ОБУЧЕНИЯ СТУДЕНТОВ ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Васильев Владимир Сергеевич[1],
канд. техн. наук,
e-mail: vsvasilev@sfu-kras.ru,

Сиротинина Наталья Юрьевна[1],
канд. техн. наук,
e-mail: nsirotinina@sfu-kras.ru,
[1]*Сибирский федеральный университет, г. Красноярск, Россия*

В статье представлен подход к проектированию программного обеспечения, дополняющий канонический процесс ICONIX этапами построения схемы базы данных, разработки макетов пользовательского интерфейса, карты диалоговых окон, описания форматов файлов и сообщений. Изменения в процесс предложены с учетом опыта, полученного при внедрении ранее созданного прототипа среды моделирования, обеспечивающего инструментальную поддержку процесса ICONIX. Основными критериями при выборе вводимых в процесс моделей являлись простота построения и наличие информации об их применении в промышленной разработке. Использование предложенного процесса направлено на повышение качества разрабатываемой спецификации требований к программному обеспечению и обеспечение более тесной ее связи с моделями, создаваемыми при проектировании. Показана возможность автоматизации поиска ряда ошибок проектирования. Из открытых источников выбраны методы проверки проектных решений, применимые к создаваемым в рамках предложенного процесса моделям, выполнение которых возможно без участия человека. Предложен ряд новых методов проверки взаимного соответствия созданных программистом моделей, специфичных для нового процесса проектирования, показана возможность их автоматизации. За счет создания и внедрения в учебный процесс инструментальных средств, обеспечивающих поддержку предложенного подхода, может быть снижена нагрузка на преподавателя и повышено качество учебных работ студентов.

Ключевые слова: проектирование программного обеспечения, язык UML, объектно-ориентированное проектирование, методология ICONIX

EXPERIENCE IN MODERNIZING THE ICONIX METHODOLOGY FOR TEACHING SOFTWARE DESIGN TO STUDENTS

Vasilev V.S.[1],
candidate of technical sciences,
e-mail: vsvasilev@sfu-kras.ru,

Sirotinina N.Y.[1],
candidate of technical sciences,
e-mail:nsirotinina@sfu-kras.ru,
[1]*Siberian Federal University, Krasnoyarsk, Russia*

The article describes an approach to software projecting that complements the canonical ICONIX process with the stages of creating a database schema, developing user interface layouts, dialog box maps, and describing file formats and messages. The changes to the process are proposed taking into account the experience gained during the implementation of a previously created prototype modeling environment that provides instrumental support for the ICONIX process. The main criteria for selecting the models introduced into the process were simplicity

of construction and availability of information about their application in industrial development. The use of the proposed process is aimed at improving the quality of the developed specification of software requirements and ensuring its closer connection with the models created during the projecting. The possibility of automating the search for a number of projecting errors is shown. Methods of verification of projecting solutions have been selected from open sources, applicable to models created within the framework of the proposed process, the implementation of which is possible without human intervention. A number of new methods for verifying the mutual correspondence of models created by the programmer, specific to the new projecting process, are proposed, and the possibility of automating them is shown. Due to creating and implementing tools in the educational process that support the proposed approach, the teacher workload can be reduced and the quality of students' academic work improved.

Keywords: software projecting, UML language, object-oriented projecting, ICONIX methodology

Введение

Одним из этапов водопадной модели разработки является проектирование, аналогичный этап присутствует в методологии RUP и в наиболее популярном в настоящий момент Scrum [1]. В ходе проектирования программного обеспечения (ПО) определяются программные объекты, модули и способ их сопряжения в соответствии с выданными заказчиком требованиями. Для этого выполняется моделирование ПО с использованием специальных языков. В работе [2] отмечается, что модель ПО дополняется на всех этапах жизненного цикла всеми участниками процесса разработки, при этом объекты и отношения модели формируются с учетом возможностей языков программирования, на которых планируется выполнять реализацию программного продукта. Модель является не целью, а ресурсом процесса разработки, для ее создания используются специальное ПО, язык и методика моделирования. В связи с этим развитие инструментальных средств, обеспечивающих моделирование ПО, является актуальным направлением исследования.

Существует множество языков моделирования и процессов проектирования, невозможно все освоить в ходе обучения в вузе. Лишь небольшое их подмножество более или менее широко применяется в промышленной разработке.

Целью настоящего исследования является модернизация методологии ICONIX для обучения студентов проектированию программного обеспечения на основе адаптации канонического процесса к производственным потребностям и расширения его инструментального сопровождения.

В рамках работы решались задачи: выявление сильных и слабых сторон подходов к проектированию ПО, применяемых в промышленной разработке; модернизация с их учетом процесса ICONIX; анализ возможности частичной автоматизации проверки создаваемых в рамках модернизированного процесса моделей ПО.

В работе применены такие теоретические и общенаучные методы исследования, как сравнительный анализ, синтез и формализация. Часть результатов подтверждена эмпирическим путем.

Обзор языков моделирования и процессов проектирования ПО приведен в **первой части статьи**. При этом внимание уделено не только использованию этих нотаций на практике, но и наличию для них специализированных программных средств, частично *автоматизирующих проверку* корректности разработанных моделей. При внедрении в учебный процесс автоматизация поиска ошибок моделирования особенно важна, так как преподаватель должен быстро и качественно проверять большое количество студенческих проектов. На наш взгляд, наиболее подходящим для обучения является процесс ICONIX, так как при его использовании студент познакомится с наиболее универсальными и часто применяемыми диаграммами UML.

Выбранный процесс проектирования был использован в учебном процессе по двум курсам, связанным с проектированием ПО на кафедре «Вычислительная техника» Сибирского федерального университета в 2019–2022 годах, а в 2020 году был создан прототип среды моделирования программного обеспечения, позволяющий строить диаграммы в предусмотренном процессом порядке. По ходу использования были выявлены сильные и слабые стороны канонического процесса ICONIX, изложенные **во второй части** статьи наряду с описанием процесса.

ICONIX – итеративный процесс, это значит, что модели многократно дорабатываются по ходу проектирования, при этом в них устраняются недочеты. Анализ результатов внедрения в учебный процесс выполнялся по оценке курсовых работ, представленных в конце обучения по предметам. **Третья часть статьи** посвящена разработке нового процесса проектирования для обучения проектированию. Предложенный процесс дополняет канонический ICONIX рядом моделей, часто применяемых в промышленной разработке ПО и позволяет повысить качество автоматизированной проверки моделей.

В литературе описан ряд методов проверки результатов проектирования, некоторые из них оказываются применимы к предложенному нами процессу. Выполнен анализ этих методов на предмет возможности проверки проекта без участия человека, результаты приведены **в четвертой части статьи**. Кроме того, предложены *новые методы* проверки моделей, специфичные для предложенного нами процесса проектирования.

1. Обзор языков моделирования и процессов проектирования ПО

Одни языки моделирования ПО позволяют выразить требования к программе, а другие – выполнить концептуальное моделирование исследуемого объекта. Модели, создаваемые при концептуальном моделировании, должны выражать сущности и отношения, поддерживаемые языками программирования и другими инструментами, поэтому языки моделирования менялись параллельно с развитием других инструментальных средств разработки ПО [3]. Ниже указаны наиболее широко применяемые на определенном этапе языки моделирования:

– моделирование программного обеспечения выполняется на основе требований к нему, которые уточняются и формализуются на начальном этапе разработки. Для этого чаще всего применяют нотацию IDEF0 или use-case диаграммы UML [4];

– базы данных являются важным элементом программных систем, а набор используемых в них сущностей и отношений отличается от используемого в программном коде. Для построения схем баз данных часто применяется ER-модель или нотация IDEF1X[1];

– так как программное обеспечение зачастую создается для встраивания в существующие бизнес-процессы, то эти процессы также моделируются перед разработкой ПО. При этом широко используются нотации UML и IDEF4 [5];

– в настоящее время наиболее распространены языки объектно-ориентированного (ОО) программирования. Для моделирования систем, разработанных с использованием ОО-подхода, был предложен ряд нотаций – OMT, OOSE и метод Буча, на их основе создан международный стандарт UML[2].

Инструментальная поддержка процесса проектирования должна обеспечивать не только возможность построения диаграмм, но и проверять их корректность и согласованность. В работе [6] приводится информация об использовании ПО для моделирования программных систем в различных секторах экономики. Отмечается, что более половины предприятий, связанных с критически важными системами, используют программу MS Visio, обладающую ограниченными возможностями UML-моделирования. Приведен ряд более развитых систем, таких как IBM – Rational Rhapsody Designer for Systems Engineers и Siemens PLM – System Modeling Workbench. Такие системы в большей степени ориентированы на прямое проектирование с использованием метода Object Oriented Analysis and Design (OOAD[3]), так как позволяют создавать модели в определенной последовательности и поддерживают более 10 различных видов диаграмм. Разнообразием используемых нотаций обусловлена высокая сложность такого подхода [7].

Решение проблемы сложности проектирования предложено в рамках процесса ICONIX [8] за счет использования *сокращенного набора наиболее универсальных диаграмм*. Для описания требований к ПО используется диаграмма прецедентов UML. На ее основе разрабатываются сначала динами-

---

[1] Нотации модели сущность-связь (ER-диаграммы). – URL: https://pro-prof.com/archives/8126 (дата обращения: 26.04.2025). – Текст: электронный.

[2] *Rumbaugh J., Jacobson I., Booch G.* The Unified Modeling Language Reference Manual. – New York: Addison-Wesley, 2021. – 568 p.; Unified Modeling Language Version 1.4.2. ISO/IEC 19501:2005. – URL: https://www.iso.org/standard/32620.html (дата обращения: 26.04.2025). – Текст: электронный.

[3] *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. – Санкт-Петербург: Невский диалект, 1998. – 558 с.

ческая, а потом статическая модели системы. Для построения этих моделей выбраны 2 диаграммы – последовательности и классов.

В ряде работ отмечается, что на сложность создания системы существенно влияет семантический разрыв между представлениями о разрабатываемой системе главных участников данного процесса [9]. Для сокращения такого разрыва в ICONIX предложено дополнить процесс этапом построения диаграмм пригодности, включающим предварительный анализ прецедентов на предмет использования в них экранных форм, сущностей и обработки ошибок.

Вышеизложенным обусловлена целесообразность применения процесса ICONIX в учебном процессе. В работе [9], помимо описания процесса, приведены часто допускаемые при проектировании ошибки, советы по их обнаружению и устранению для каждого этапа проектирования.

2. Канонический процесс ICONIX. Слабые и сильные стороны

ICONIX – итеративный процесс, основанный на прецедентах, заключается в следующем:

1) на основе сведений, получаемых от заказчика, формируется набор прецедентов, строится use-case диаграмма UML и дополняется текстовым описанием;

2) для каждого прецедента выполняется:

a) разработка диаграммы пригодности. Анализируется текстовое описание, множество связанных с прецедентом элементов предметной области разделяется на 3 части – сущности, процессы и граничные объекты. Изображаются зависимости между элементами модели;

b) разработка динамической модели. Текстовое описание прецедента (последовательность действий) изображается с использованием нотации диаграммы последовательности UML. Зависимости между элементами модели изображаются в виде передачи сообщений между объектами, эквивалентной вызову функций в исходном коде программ;

c) разработка статической модели. За основу может быть взята модель предметной области, которая может разрабатываться с использованием нотации диаграммы классов UML. Эта модель дополняется элементами диаграмм последовательности, – проверяется наличие класса для каждого объекта каждой последовательности, а также соответствие методов классов и сообщений на диаграммах последовательности.

После каждого этапа выполняется проверка согласованности результатов с построенными ранее моделями. Выявление и устранение в ходе проектирования ошибок, допущенных на предыдущих этапах, считается нормальным явлением, в этом заключается итеративность процесса. За счет этого повышаются качество и надежность проекта, например, при разработке диаграммы пригодности не всегда можно сразу выявить все необходимые сущности, но потребность в них становится очевидной при построении диаграмм последовательности.

В результате проектирования формируются модели, близкие к коду программы, в ICONIX учитывается, что программа будет объектно-ориентированной. Однако изначально у проектировщика имеются лишь слабо формализованные требования заказчика. В международном стандарте прецеденты являются одним из способов описания функциональных требований к ПО[4]. Целесообразно формировать их на начальном этапе, а затем использовать при разработке. Прецеденты анализируются отдельно друг от друга, но результаты могут соединяться в одной диаграмме классов. Это значит, что при формировании прецедентов уже выполняется декомпозиция планируемого объема работ и снижается сложность дальнейшего проектирования.

Добавление этапа анализа пригодности не только упрощает переход от прецедентов, записанных в достаточно свободной форме, к диаграммам последовательности, выполненным по строгим правилам, но также подталкивает к использованию шаблона проектирования Модель – Вид – Контроллер (MVC). Правила выполнения диаграмм пригодности запрещают напрямую связывать модель (сущность) и вид (граничный объект), это приводит к изоляции модели от вида, а значит – улучшению инкапсуляции модулей.

[4] Systems and software engineering – Life cycle processes – Requirements engineering. ISO/IEC/IEEE 29148:2018(E) International. – URL: https://www.iso.org/standard/72089.html (дата обращения: 26.04.2025). – Текст: электронный.

Ряд недостатков и особенностей использования был выявлен в результате внедрения описанного подхода к проектированию в учебный процесс Сибирского федерального университета. С 2019 года студенты применяли его при разработке информационных систем (ИС). Любая такая система работает с базой данных (БД) и реализует пользовательский интерфейс.

На этапе анализа пригодности БД определяется как сущность, а при дальнейшем проектировании формируется ее интерфейс (набор публичных функций). При реализации системы студенты чаще всего использовали язык запросов SQL или хранили данные в файлах, поэтому выделенная сущность реализовывалась с помощью шаблонов проектирования *Фасад* или *Адаптер*. Проблема заключается в том, что ни в ICONIX, ни в OOAD не уделяется внимание проектированию БД.

Задача спецификации сущностей возникает также при разработке клиент-серверных приложений. Роль интерфейса сервера играет API, наиболее популярным вариантом его описания является стандарт OpenAPI/Swagger[5]. При его использовании возможно сгенерировать обертки API для разных языков программирования и документацию. Для спецификации форматов файлов или сообщений удобно использовать схемы, например, JSON или XML.

Текстовое описание любого прецедента для ИС сводится к последовательности действий, которые выполняет пользователь. При этом ключевую роль играет пользовательский интерфейс, так как именно через него пользователь взаимодействует с системой. Элементы пользовательского интерфейса (диалоговые окна) выделятся при выполнении анализа пригодности, однако интерфейс должен быть проработан раньше. При разработке спецификации требований ИС со сложной структурой окон до 50 % объема текстового описания прецедентов может занимать навигация между экранными формами.

Был **реализован и опробован в учебном процессе прототип** среды моделирования, позволяющий разрабатывать диаграммы в соответствии с процессом ICONIX. Модели описывались с помощью текста в формате PlantUML[6], на его основе с использованием существующих инструментов генерировалась графическая форма представления моделей. Выявлена возможность частичной автоматизации проверки проектных решений, созданных студентами.

В прототипе поддерживается небольшой набор алгоритмов проверки проектных решений. Предупреждения выдаются в следующих случаях:

1) для какого-либо прецедента не созданы диаграммы пригодности и последовательности;

2) на диаграммах пригодности и последовательности используются запрещенные виды связей, например, актор напрямую соединен с сущностью;

3) на диаграмме классов присутствует сущность, не использованная ни в одном прецеденте в проекте, то есть не представленная на диаграммах последовательности;

4) на диаграммах используется более 8 ключевых объектов (диаграмма перегружена).

Часть проверок выполняется PlantUML, например, в нем невозможно изобразить на диаграмме последовательности сообщения, отправленные в прошлое или использовать на ней неправильные типы связей (каждому типу сообщений соответствует свой вид соединительной линии).

Результаты внедрения прототипа докладывались на конференциях[7]. Были проанализированы курсовые работы студентов, выполненные без специальных инструментальных средств, на предмет наличия ошибок разных видов. Например, в 10 % процентах работ на диаграмме последовательности используются неправильные типы связей. При использовании прототипа такие ошибки в работах студентов не встречаются, а преподаватель не тратит время на их поиск и может сфокусироваться на других аспектах работы.

На наш взгляд, реализация таких инструментальных средств окажет положительное влияние на учебный процесс, так как студенты получат возможность проверять свои проекты без участия препо-

[5] OpenAPI Specification v3.1.0. – URL: https://spec.openapis.org/oas/v3.1.0 (дата обращения: 26.04.2025). – Текст: электронный.
[6] PlantUML language reference guide. –URL: https://plantuml.com/ru/guide (дата обращения: 26.04.2025). – Текст: электронный.
[7] *Шишкина И.С., Исайкин А.А., Хантимиров А.Г.* Методы поиска ошибок проектирования для учебной среды объектно-ориентированного моделирования // Наука в современном мире: результаты исследований и открытий: сборник научных трудов по материалам IV Международной научно-практической конференции. – Анапа, 2022. – С. 110–116; *Артемьев Л.С.* Учебная среда объектно-ориентированного моделирования в обеспечении дистанционного образовательного процесса // Проспект Свободный – 2023: материалы XIX Международной научной конференции студентов, аспирантов и молодых ученых. – Красноярск, 2023. – С. 1382–1384.

давателя, а качество проверки работ повысится. Решено внести изменения в канонический процесс, чтобы устранить обнаруженные недостатки.

3. Модернизированный процесс ICONIX

Функциональные требования к ПО в процессе ICONIX описываются в виде прецедентов. Прецедент соответствует функции системы, важной для заказчика и задается в форме диалога пользователя системой. Для взаимодействия с пользователем система предоставляет пользовательский интерфейс. В связи с этим разработку макетов пользовательского интерфейса предложено вести параллельно с описанием прецедентов. При этом для каждого прецедента определяется набор граничных объектов, за каждым из которых закрепляется изображение, представляющее макет интерфейса.

При проектировании клиент-серверных приложений каждый запрос к серверу можно рассматривать как прецедент, а вместо макета интерфейса использовать схему запроса или любой другой текстовый или графический формат описания способа взаимодействия.

Для описания переходов между окнами решено использовать нестандартизированную диаграмму потока экранов, описанную М. Фаулером[8]. Аналогичный подход описан К. Вигерсом как карта диалоговых окон[9]. Для создания диаграммы можно использовать диаграмму состояний UML, представляя окна состояниями, а возможность перехода между ними – дугами.

Рассматривались различные варианты валидации проектов. Часть существующих аналогичных решений посвящена интерпретации UML-моделей. При этом выбираются конкретные виды поведенческих диаграмм, которые должны детально прорабатываться, и дополняются *ограничениями* [10]. Такой подход позволяет верифицировать системы определенных типов, но непригоден для проверки моделей, созданных при прямом проектировании произвольных программных систем. В ряде случаев для задания ограничений используются свои собственные языки [11; 12], а в других – объектный язык ограничений (OCL), являющийся частью стандарта UML. В работе [11] описан ряд инструментальных средств, позволяющих выполнять проверку соответствия системы ограничениям на OCL. Задание ограничений позволяет повысить качество проверки моделей, но существенно повышает сложность проектирования, поэтому принято решение отказаться от их поддержки.

Предложено дополнить процесс этапом описания форматов сущностей. Форматы данных являются элементом спецификации требований, поэтому должны специфицироваться параллельно с проработкой прецедентов. По ходу проектирования они могут уточняться, так как процесс является итеративным. Отдельная сущность описывается в текстовом формате, это может быть схема документа или, например, форма Бэкуса-Наура. Для сущностей, представленных таблицами базы данных, можно использовать ER-диаграмму, при этом на одной ER-диаграмме располагается сразу несколько сущностей. Создание подобных моделей поддерживается инструментами PlantUML.

Возможна частичная генерация диаграмм по ранее созданным моделям. Часть объектов на диаграмму последовательности могут быть автоматически перенесены с диаграммы пригодности. Предусловия могут быть заимствованы из текстового описания прецедентов. Диаграмма классов может быть частично сгенерирована по диаграммам последовательности. Часть элементов на эти диаграммы может быть добавлена автоматически на основе существующих связей прецедента с макетами интерфейса (граничными объектами) и сущностями.

Не рекомендуется тратить на разработку диаграмм пригодности слишком много сил, так как они являются временными. После проработки классов предлагается удалить диаграммы пригодности, чтобы не тратить силы на приведение их в согласованное состояние с остальным проектом.

Более 90 % прецедентов должны иметь альтернативные последовательности [13], обычно их описание в значительной мере дублирует основную последовательность. Создание отдельных диаграмм последовательности для каждой альтернативы не только требует дополнительных усилий и времени, но также ведет к усложнению проекта. Поэтому в создаваемой системе поддерживается лишь одна диа-

[8] *Fowler M.* UML distilled: a brief guide to the standard object modeling language. – New York: Addison-Wesley, 2018. – 208 p.
[9] *Wiegers K.E., Beatty J.* Software requirements. – London: Pearson Education, 2013. – 620 p.

грамма последовательности для каждого прецедента, альтернативные варианты изображаются с помощью блоков *alt*, предусмотренных стандартом UML.

На рисунке 1 последовательность моделирования наглядно представлена в нотации диаграммы активности UML, ниже приведено более подробное описание каждого шага:

1) в ходе формирования спецификации требований вместе с заказчиком (преподавателем) выявляются роли пользователей, работающих в системе. Создается основа модели предметной области (в нотации диаграммы классов);

2) для каждой роли формируется набор прецедентов;

3) для каждого прецедента разрабатываются:

a) макеты интерфейса (каждое окно соответствует одному граничному объекту);

b) сущности и текстовое описание форматов сущностей, новые выявленные сущности вносятся в модель предметной области;

c) текстовое описание прецедента;

d) диаграмма пригодности (на основе сгенерированного шаблона, содержащего граничные объекты и сущности, прикрепленные к прецеденту);

e) диаграмма последовательности (на основе шаблона, содержащего объекты соответствующей диаграммы пригодности);

4) на основе ER-диаграммы и набора диаграмм последовательности строится диаграмма классов;

5) диаграммы пригодности «выкидываются», появляется возможность их скрытия, но в любом случае они не используются при выполнении дальнейшего анализа.

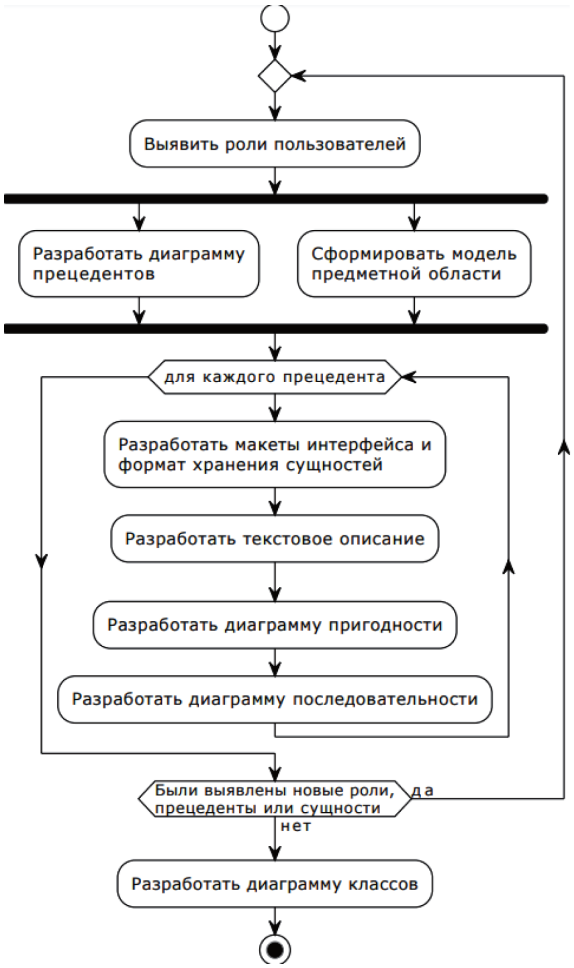

Рисунок 1 – Предлагаемая последовательность моделирования[10]

[10] Составлено авторами.

Начальные этапы не обязательно сразу выполнять очень тщательно, допущенные недочеты с большой вероятностью выявятся на следующих этапах. При обнаружении ошибки необходимо повторить процесс с этапа, на котором она была допущена. С другой стороны, допустить ошибку на заключительных этапах сложнее, так как создаваемые модели сверяются с разработанными ранее и даже частично создаются на их основе.

4. Методы выявления ошибок проектирования

Для процесса ICONIX был выделен ряд «типичных ошибок проектирования» [13], допускаемых при разработке каждого вида диаграммы. Часть из них возможно выявить без участия человека. Ниже приведены предложенные на их основе правила проверки моделей, а также ряд правил, использующих особенности предложенного учебного процесса объектно-ориентированного моделирования.

1. Не проверяют наличие в тексте прецедентов ссылок на объекты предметной области.

До этапа текстового описания с прецедентом связываются граничные объекты и сущности. Рекомендуется давать им характерные несклоняемые имена, например: User, StudentsDatabase, AuthForm. Тогда проверка соблюдения правила сводится к поиску в тексте прецедента соответствующих слов.

2. Не проверяют, все ли альтернативные последовательности действий были рассмотрены.

При отсутствии альтернативной последовательности предлагается выводить предупреждение, так как примерно у 10 % прецедентов они действительно могут отсутствовать. Предупреждение должно содержать рекомендации о необходимости задать к каждому предложению текстового описания вопрос: «Что в прецеденте могло пойти не так?». Возможность явного задания и хранения основной и альтернативных последовательностей была реализована уже в прототипе учебной среды ОО-моделирования.

3. Создают длинные текстовые описания прецедентов. Включают слишком много или слишком мало контроллеров в диаграмму пригодности.

Возможно ввести ограничения на длину текстового описания, так, Розенберг пишет: «Главная последовательность прецедента не должна быть длиннее одного-двух абзацев. Каждая альтернативная последовательность должна состоять из одного-двух предложений».

С другой стороны, диаграмма пригодности «длинного» прецедента обязательно будет содержать слишком большое количество контроллеров. Предлагается выводить предупреждение с предложением разбить сложный прецедент на несколько более простых, если число контроллеров оказалось более 7. Связаны такие ограничения с предельным объемом информации, которая может быть быстро усвоена человеком («число Миллера»).

Прецедент, содержащий единственный контроллер, скорее всего, также не имеет смысла.

4. При разработке диаграмм пригодности нарушают паттерн «граничный объект – контроллер – сущностный объект».

Диаграммы пригодности должны использовать паттерн «граничный объект – контроллер – сущностный объект», однако достаточно проверить наличие в диаграмме пригодности «запрещенных» связей: «граничный объект – сущность», «актор – контроллер», «актор – сущность». Стоит проверять, что все граничные объекты диаграммы связаны с актором.

5. Не отражают альтернативные последовательности на диаграммах пригодности.

Рекомендуется раскрашивать красным цветом элементы диаграмм пригодности и последовательности, имеющие отношение к альтернативным вариантам. Возможно проверять их наличие для прецедентов, имеющих хотя бы одну альтернативную последовательность, при отсутствии выдавать предупреждение.

6. Не задумываются о том, в каком направлении рисовать стрелку на диаграмме последовательности. Не проверяют согласованность имен, использованных в диаграммах последовательности и классов.

Каждый объект диаграммы последовательности относится к какому-либо классу, поэтому он должен быть представлен на соответствующей диаграмме. Стрелки (сообщения) диаграммы последо-

вательности должны соответствовать методам классов. Предлагается проверять наличие скобок (соответствующих вызову функции) в тексте сообщения.

7. Не обновляют модель предметной области. Не проверяют ее соответствие с диаграммой классов.

Диаграмму классов предлагается строить на основе модели предметной области. Однако несоответствия могут появиться из-за итеративности предложенного процесса. Предлагается выполнять проверку наличия на диаграмме классов всех элементов предметной области и их атрибутов.

8. Не занимаются поиском атрибутов классов из предметной области.

Значительная часть атрибутов должна соответствовать элементам граничных объектов или использоваться при описании форматов данных/сообщений.

При обнаружении полей структур, не представленных в соответствующих классах предметной области, предлагается выдавать предупреждение, так как в ряде случаев в сообщениях передается служебная (вычисляемая) информация, не имеющая прямого отношения к сущности.

9. Составляют диаграммы последовательности не для каждого прецедента. Не учитывают альтернативные варианты в диаграммах последовательности.

Предлагается проверять наличие диаграмм последовательностей для каждого прецедента и альтернативных блоков в нем для каждого альтернативного варианта.

10. Не соблюдают правила выполнения отдельных видов диаграмм.

Проверка отсутствия на диаграммах недопустимых типов связей и объектов. Например, на диаграмме классов не могут размещаться прецеденты, а на диаграмме прецедентов – классы; между ролью и прецедентом допустимо только отношение ассоциации; между прецедентами – отношения включения, расширения и обобщения.

Инициатором действий в каждой диаграмме последовательности должен быть актор или объект, получивший «найденное сообщение». Перед отправкой своего первого сообщения каждый объект на диаграмме последовательности должен получить хотя бы одно сообщение.

11. Разрабатывают диаграммы последовательности, пригодности, модели предметной области, не соответствующие друг другу.

На диаграмме пригодности, построенной для некоторого прецедента, должны использоваться только граничные объекты, соответствующие макетам интерфейса, связанные с этим прецедентом, а также только сущности, связанные с этим прецедентом. С другой стороны, все привязанные к прецеденту сущности должны быть обязательно использованы на диаграмме пригодности.

На диаграммах последовательности, построенных для прецедента, должны быть использованы сущности и граничные объекты с соответствующих диаграмм пригодности.

Если на диаграмме пригодности граничный объект и сущность взаимодействуют через процесс, значит, на диаграмме последовательности должен быть набор связывающих их сообщений, но они не обязательно должны связываться сообщением напрямую, так как процесс может быть заменен новым классом.

Заключение

Предложенный процесс проектирования предусматривает: параллельную разработку прецедентов, макетов интерфейса, форматов сущностей и карты диалоговых окон; создание ER-диаграммы базы данных; спецификацию API-системы с использованием стандарта OpenAPI. Эти изменения положительно скажутся на качестве подготовки специалистов, так как вводимые подходы и технологии широко применяются в промышленной разработке ПО.

При использовании предложенного процесса к проектированию оказываются применимы все методы проверки проектных решений, описанные для процесса ICONIX. Выделено подмножество методов, для которых возможна программная реализация. За счет введения в процесс новых этапов и моделей ПО стали возможны и предложены новые методы проверки проектных решений. Инструментальная поддержка описанного процесса обеспечит не только снижение нагрузки на преподавателя, но

также повысит скорость и качество проверки работ, позволит студентам выявлять часть ошибок без участия преподавателя.

Планируется реализация инструментальных средств поддержки предложенного процесса проектирования, оценка результатов внедрения в учебный процесс, в том числе при выполнении выпускных квалификационных работ.

Список литературы

1. *Shafiee S., Wautelet Y., Hvam L., Sandrin E., Forza C.* Scrum versus Rational Unified Process in facing the main challenges of product configuration systems development // Journal of Systems and Software. – 2020. – Vol. 170. – P. 110732.
2. *Lucas F.J., Molina F., Toval A.* A systematic review of UML model consistency management // Information and Software technology. – 2009. – Vol. 51, No. 12. – P. 1631–1645.
3. *Henderson-Sellers B., Gonzalez-Perez C., Eriksson O., Agerfalk P.J., Walkerden G.* Software modelling languages: A wish list // 2015 IEEE/ACM 7th International Workshop on Modeling in Software Engineering. – 2015. – P. 72–77.
4. *Marca D.A.* SADT/IDEF0 for augmenting UML, agile and usability engineering methods // Software and Data Technologies: 6th International Conference. – 2013. – Vol. 3. – P. 38–55.
5. *Fu M., Wang D.D., Wang J., Li M.* UML vs IDEF: Modeling Method of Operational Task Combined with IDEF and UML // 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference. – 2018. – P. 1443–1447.
6. *Романов А.А., Шпотя Д.А.* Преодоление недостатков программно-методического инструментария модельно-ориентированного системного инжиниринга, используемого при проектировании систем // Известия Самарского научного центра Российской академии наук. – 2020. – № 22 (6). – С. 92–103.
7. *Самонов А.В., Малышев С.Р., Краснов С.В., Савкин А.Л.* Методы и средства реализации процесса сквозного контроля качества артефактов жизненного цикла разработки критически важных информационных систем // Автоматизация процессов управления. – 2019. – № 3. – С. 12–20.
8. *Ma Q.* System requirements analysis with ICONIX process case study: group project peer-assessment tool // Journal of Business Cases and Applications. – 2019. – Vol. 21. – P. 1–21.
9. *Wang B., Rosenberg D., Boehm B.W.* Rapid realization of executable domain models via automatic code generation // 2017 IEEE 28th Annual Software Technology Conference (STC). – IEEE. – 2017. – P. 1–6.
10. *Shen W., Compton K., Huggins J.* A UML validation toolset based on abstract state machines // International Conference on Automated Software Engineering. – 2001. – P. 315–318.
11. *Ober I., Graf S., Ober I.* Validating timed UML models by simulation and verification // International Journal on Software Tools for Technology Transfer. – 2006. – Vol. 8, No. 2. – P. 128-145.
12. *Akehurst D.H., Howells G., McDonald-Maier K.D.* Supporting OCL as part of a Family of Languages // Proceedings of the MoDELS. – 2005. – Vol. 5. – P. 30–37.
13. *Rosenberg D., Stephens M., Collins-Cope M.* ICONIX Process: A Core UML Subset // Agile Development with ICONIX Process: People, Process, and Pragmatism. – 2005. – P. 39–59.

References

1. *Shafiee S., Wautelet Y., Hvam L., Sandrin E., Forza C.* Scrum versus Rational Unified Process in facing the main challenges of product configuration systems development // Journal of Systems and Software. – 2020. – Vol. 170. – P. 110732.
2. *Lucas F.J., Molina F., Toval A.* A systematic review of UML model consistency management // Information and Software technology. – 2009. – Vol. 51, No. 12. – P. 1631–1645.
3. *Henderson-Sellers B., Gonzalez-Perez C., Eriksson O., Agerfalk P.J., Walkerden G.* Software modelling languages: A wish list // 2015 IEEE/ACM 7th International Workshop on Modeling in Software Engineering. – 2015. – P. 72–77.
4. *Marca D.A.* SADT/IDEF0 for augmenting UML, agile and usability engineering methods // Software and Data Technologies: 6th International Conference. – 2013. – Vol. 3. – P. 38–55.

5. *Fu M., Wang D.D., Wang J., Li M.* UML vs IDEF: Modeling Method of Operational Task Combined with IDEF and UML // 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference. – 2018. – P. 1443–1447.
6. *Romanov A.A., Shpotya D.A.* Preodolenie nedostatkov programmno-metodicheskogo instrumentariya model'no-orientirovannogo sistemnogo inzhiniringa, ispol'zuemogo pri proektirovanii sistem // Izvestiya Samarskogo nauchnogo centra Rossijskoj akademii nauk. – 2020. – № 22 (6). – S. 92–103.
7. *Samonov A.V., Malyshev S.R., Krasnov S.V., Savkin A.L.* Metody i sredstva realizacii processa skvoznogo kontrolya kachestva artefaktov zhiznennogo cikla razrabotki kriticheski vazhnyh informacionnyh sistem // Avtomatizaciya processov upravleniya. – 2019. – № 3. – S. 12–20.
8. *Ma Q.* System requirements analysis with ICONIX process case study: group project peer-assessment tool // Journal of Business Cases and Applications. – 2019. – Vol. 21. – P. 1–21.
9. *Wang B., Rosenberg D., Boehm B.W.* Rapid realization of executable domain models via automatic code generation // 2017 IEEE 28th Annual Software Technology Conference (STC). – IEEE. – 2017. – P. 1–6.
10. *Shen W., Compton K., Huggins J.* A UML validation toolset based on abstract state machines // International Conference on Automated Software Engineering. – 2001. – P. 315–318.
11. *Ober I., Graf S., Ober I.* Validating timed UML models by simulation and verification // International Journal on Software Tools for Technology Transfer. – 2006. – Vol. 8, No. 2. – P. 128 145.
12. *Akehurst D.H., Howells G., McDonald-Maier K.D.* Supporting OCL as part of a Family of Languages // Proceedings of the MoDELS. – 2005. – Vol. 5. – P. 30–37.
13. *Rosenberg D., Stephens M., Collins-Cope M.* ICONIX Process: A Core UML Subset // Agile Development with ICONIX Process: People, Process, and Pragmatism. – 2005. – P. 39–59.

Статья поступила в редакцию: 26.12.2024
Статья принята к публикации: 24.03.2025

Received: 26.12.2024
Accepted: 24.03.2025