УДК 004

РАЗРАБОТКА ЭКСПЕРТНОЙ СИСТЕМЫ ПО ПОСТРОЕНИЮ АРХИТЕКТУРЫ ПРОГРАММНЫХ ПРОДУКТОВ

А. Е. Гришин[1] [0000-0002-7355-4878], К. А. Григорян[2] [0000-0001-6470-1832]

[1, 2]Казанский (Приволжский) федеральный университет, ул. Кремлевская, 35, г. Казань, 420008

[1]andrey.grishin.work@gmail.com, [2]karigri@yandex.ru

Аннотация

Статья посвящена автоматизации этапа проектирования программного обеспечения. Проанализированы причины высокого значения данного этапа и актуальность его автоматизации. Рассмотрены основные стадии названного этапа и существующие системы, позволяющие автоматизировать каждую из них. Предложено собственное решение в рамках задачи рефакторинга структуры классов на основе метода комбинаторной оптимизации. Разработан и протестирован на реальной модели метод решения, позволяющий улучшить качество иерархии классов.

***Ключевые слова**: автоматизация, проектирование, рефакторинг, архитектура ПО, ООП, оптимизация*

ВВЕДЕНИЕ

В последние несколько лет наблюдается стремительный рост объема рынка разработки программного обеспечения (ПО) [1], увеличивается количество разрабатываемых программных продуктов (ПП) и повышается спрос на квалифицированных специалистов в этой области. В условиях большого объема работы и ограниченности трудовых ресурсов оптимизация экономической эффективности процесса разработки становится необходимостью.

Одним из наиболее важных в процессе разработки ПО является этап проектирования, который представляет собой формализованное описание внутренних и внешних свойств разрабатываемой системы, а также того, как система взаимодействует с внешними компонентами и конечными пользователями [2]. Вместе с

языком программирования, фреймворком, базой данных и другими программными компонентами все вышеперечисленное составляет архитектуру ПО.

Значимость проектирования заключается в том, что на основе решений, принятых на этом этапе, осуществляется вся последующая разработка [3, 4]. Соответственно, при качественно проведенном проектировании можно получить следующие организационные преимущества:

- правильную оценку времени и стоимости разработки;
- увеличение вероятности избежать дополнительных доработок и согласований требований;
- возможность избежать неудовлетворенности конечным результатом и разногласий между заказчиком и исполнителем.

Кроме того, правильно спроектированная архитектура позволяет добиться и технических преимуществ:

- снижения скорости накопления технического долга;
- ускорения процесса разработки;
- уменьшения вероятности возникновения дефектов.

В каждом из перечисленных пунктов можно достичь и обратного эффекта, если совершить большое количество ошибок на данном этапе или же вовсе пренебречь им [3, 4]. Суммарно все полученные негативные эффекты будут выражаться в увеличении сроков и стоимости разработки, что всегда является крайне нежелательным.

Сам по себе этап проектирования является затратным в плане экономических и временных ресурсов, так как, в зависимости от размера разрабатываемой системы, его продолжительность может варьироваться от коротких до очень длительных сроков [5]. Также, ввиду высокой цены ошибки на данном этапе, предъявляются строгие требования к квалификации специалистов, вовлеченных в него [5]. Перспектива полной или частичной автоматизации этого процесса позволит сократить издержки и снизить влияние человеческого фактора на конечные результаты. Таким образом, на сегодняшний день задача автоматизации проектирования является крайне актуальной.

ОБЗОР ЛИТЕРАТУРЫ

Перед чем начать рассматривать аналоги, описанные в литературе, определим, из чего состоит процесс проектирования. Упрощенно его можно разделить на три основных этапа: анализ требований, непосредственно разработка архитектуры и преобразование полученных теоретических программных компонентов в реальные.

Анализ требований традиционно выделяют в качестве отдельного этапа жизненного цикла ПО. Однако он неразрывно связан и с проектированием, ведь именно на основе бизнес-требований формируются программные компоненты и их связи друг с другом, а также модели архитектуры системы, технологии разработки и реализации. Рассмотрим системы, позволяющие автоматизировать этап анализа требований.

TOVE (TOronto Virtual Enterprise) [6]. Этот проект позволяет создавать универсальные и повторно используемые корпоративные модели данных. Авторы разработали несколько адаптивных онтологий, которые можно использовать для собственных проектов, в зависимости от предметной области. Каждая онтология оснащена поддержкой общей терминологии предметной области с определениями каждого термина, поддержкой семантики в наборе аксиом, позволяющей автоматически генерировать связи между компонентами в проекте, а также многими другими характеристиками. Реализован данный программный комплекс в большей степени на языке Prolog.

Marrying Ontology and Software Technologies (MOST) [7]. Проект направлен на улучшение качества разработки программного обеспечения с помощью использования онтологий и технологии рассуждения. Для достижения этой цели авторы планируют реализовать бесшовную интеграцию технологии онтологий в разработку ПО на основе моделей MDSD. В результате разработка будет вестись на основе онтологий, что в свою очередь ускорит процесс проектирования и снизит вероятность несогласованностей в требованиях. В данный момент проект находится на стадии реализации.

KOntoR [8]. Этот проект реализует основанный на онтологии подход к повторному использованию программного обеспечения. Авторы задались целью

при помощи базовых знаний, представленных в виде онтологий, повысить ценность повторно используемых библиотек. Это было достигнуто путем семантической интеграции явных и неявных метаданных, что обеспечило средства для получения новых фактов. Иными словами, при возникновении требований, которые ранее уже были проанализированы и для которых уже был подобран соответствующий программный комплекс, система будет автоматически связывать их с уже разработанными решениями.

SEON: The Software Engineering Ontology Network [9]. Данный проект фактически является хорошо обоснованной сетью эталонных онтологий и механизмов для получения и включения в сеть новых интегрированных онтологий предметной области. Благодаря этому достигается возможность многократного использования этих онтологий в качестве шаблонов для других.

Заключительный этап проектирования — преобразование полученных формальных структур в реальные программные компоненты, то есть кодогенерация. В отличие от предыдущих этапов данная стадия, как правило, не требует высокого уровня квалификации, абстрактного мышления или практических навыков в области проектирования архитектуры ПО. Она представляет собой преимущественно реализацию уже разработанного проекта на том или ином технологическом стеке. В результате формализованности данного этапа он наиболее часто подвергается автоматизации. Рассмотрим системы, позволяющие автоматизировать преобразования формально описанных структур в реальные программные компоненты.

UML/Code Generation Tool [10]. Является инструментом для генерации кода из унифицированного языка моделирования UML, который работает на Windows, Linux и MacOS X. Этот инструмент дает средство моделирования, которое включает диаграммы UML, такие как диаграммы вариантов использования, классов, последовательностей, связей. Помимо этого, он позволяет генерировать код на C++, Java, Idl, PHP, Python и MySQL или импортировать код в диаграммы. Последняя его версия была выпущена в июле 2018 года.

BPwin [11]. Это программный продукт, разработанный компанией ltd. Logic Works. Он предназначен для поддержки процесса создания информационных систем. Относится к категории CASE средств верхнего уровня. Данный инструмент

является достаточно развитым средством моделирования, позволяющим проводить анализ, документирование и улучшение бизнес-процессов. С его помощью можно моделировать действия в процессах, определять их порядок и необходимые для них ресурсы. Модели BPwin создают структуру, необходимую для понимания бизнес-процессов, выявления управляющих событий и порядка взаимодействия элементов процесса между собой.

Erwin Data Modeler [11]. Это программное обеспечение для проектирования и документирования баз данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологии баз данных и среды развертывания. По своей сути Erwin является CASE-средством. Пользователи могут использовать Erwin Data Modeler как способ создания концептуальной модели данных или создания логической модели, не зависящей от конкретной технологии базы данных. В дальнейшем эта схематическая модель может быть использована для создания физической модели данных.

Design/IDEF [12]. Это CASE-пакет, который по заверениям авторов автоматизирует многие этапы проектирования сложных систем различного назначения: формулировку требований и целей проектирования, разработку спецификаций, определение компонентов и взаимодействий между ними, документирование проекта, проверку его полноты и непротиворечивости. Наиболее успешно пакет применяется для описания и анализа деятельности предприятия. Он позволяет оценить такую структуру, как единую сущность, сочетающую в себе управленческие, производственные и информационные процессы. В основе пакета лежит методология структурного проектирования и анализа сложных систем IDEF0/SADT.

Silverrun [13]. Данное CASE-средство американской фирмы Computer Systems Advisers, Inc. используется для анализа и проектирования больших информационных систем и ориентировано в большей степени на спиральную модель жизненного цикла. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм «сущность–связь»).

Rational Rose [14]. Данное ПО является еще одним CASE-средством проектирования и разработки информационных систем и программного обеспечения для управления предприятиями. Как и другие вышеперечисленные CASE-средства, его можно применять для анализа и моделирования бизнес-процессов. Принципиальное отличие Rational Rose от других средств заключается в объектно-ориентированном подходе. Графические модели, создаваемые с его помощью, основаны на объектно-ориентированных принципах и языке UML. Данный инструмент моделирования позволяет разработчикам создавать целостную архитектуру процессов предприятия, сохраняя все взаимосвязи и управляющие методы между различными уровнями иерархии.

Vantage Team Builder [15]. Этот интегрированный программный продукт ориентирован на реализацию каскадной модели жизненного цикла ПО. В качестве отличительных особенностей этой системы можно выделить возможность программирования на языке C со встроенным SQL и многопользовательский доступ к репозиторию проекта, который осуществляется за счет возможности работы приложения в конфигурации «клиент–сервер».

Нетрудно заметить, что среди аналогов не представлены решения, позволяющие автоматизировать основной этап проектирования, а именно, разработку архитектуры. На данный момент не существует систем, позволяющих автоматизировать ее непосредственную разработку в той же мере, в которой ее выполняет соответствующий специалист.

МЕТОДОЛОГИЯ

В чем же заключаются причины проблем автоматизации проектирования? Деятельность в этом направлении ведется уже более сорока лет [16], и были достигнуты определенные успехи в этой области, описанные в предыдущей главе. Однако проблема так и не была решена в полной мере: на текущий момент, основываясь только на бизнес-требованиях, нельзя получить архитектурную модель, готовую к дальнейшей разработке. Происходит это по разным причинам.

Во-первых, разработка качественной архитектурной модели невозможна без нетривиальных знаний и опыта программного архитектора, а именно, понимания того, какие из требований нуждаются в соответствующих программных решениях. Проблема заключается в том, что разная совокупность таких требований может

быть связана с разными программными компонентами, и попытка перебрать все возможные комбинации будет приводить к комбинаторному взрыву. Кроме того, сами выбранные программные компоненты должны корректно взаимодействовать между собой. И плюс ко всему – программные компоненты имеют разный уровень представления, например, имеется ряд баз данных, для каждой из которых существует множество драйверов для множества языков программирования, в каждом из которых существуют свои паттерны представления данных и обеспечения доступа к ним. Выбор каждого из этих пунктов должен быть аргументирован наличием соответствующих бизнес-требований.

Во-вторых, уже на текущий момент развития отрасли разработки программного обеспечения количество программных компонентов и подходов к ним достигло огромных масштабов. Понятие разработки постоянно делится на специализации, каждая из которых имеет свою тенденцию развития с точки зрения как программной, так и методологической частей. Даже просто учесть их все и структурировать между собой представляется сложной задачей.

Третья проблема заключается в скорости развития технологий. Теоретически, решив две проблемы, описанные выше, и начав разработку подобной системы на основе текущей ситуации в отрасли, по ее окончании можно с высокой вероятностью получить неактуальный результат. Причина этого – высокая скорость появления на рынке новых программных решений, некоторые из которых могут изменять саму концепцию разработки в той или иной области. Таким образом, встает задача постоянного встраивания новых решений в уже разработанную систему.

На основе проведенного исследования мы приняли решение провести разработку в такой области этапа проектирования, как рефакторинг структуры классов. Подавляющее большинство достаточно больших систем программного обеспечения реализуется с помощью подхода ООП [17]. Если говорить о его преимуществах, то следует выделить гибкость, экономию времени разработчиков по мере развития системы и сокращение общей кодовой базы, что не позволяет проекту засоряться. Из недостатков имеет смысл сказать о достаточно сложном старте разработки в сравнении с процедурным подходом и некоторое снижение производительности работы программы. Однако следует сделать оговорку, что в

последнее время набирает популярность микросервисная архитектура [18], в основе которой часто используют языки с меньшим уровнем абстракций. Тем не менее вопрос проектирования качественной иерархии классов все еще актуален.

Однако даже такая задача является достаточно сложной, ведь задачи объектно-ориентированного проектирования часто могут быть противоречивыми и зависеть от контекста. Решаются они на основе опыта программного архитектора и его понимания бизнес-задачи. Однако существует определенный класс свойств объектно-ориентированной иерархии, характерных для подавляющего большинства систем. Набор таких свойств возьмем из работы [19]:

- количество абстрактных суперклассов;
- количество повторяющихся методов;
- количество неиспользуемых методов;
- количество безликих классов.

Именно эти свойства были выбраны потому, что они являются максимально непротиворечивыми, а также измеримыми. Однако при желании ничего не мешает добавить дополнительные свойства или же изменить существующие.

Далее необходимо определить, с помощью каких действий будет происходить рефакторинг архитектуры. Сохраняя баланс между эффективностью и реализуемостью, мы выбрали следующие действия:

- Перемещение методов по иерархии;
  - переместить метод в суперкласс;
  - переместить метод в дочерний класс;
- Управление подклассами;
  - извлечь подкласс из суперкласса;
  - объединить подкласс с суперклассом;
- Управление абстракциями;
  - сделать класс абстрактным;
  - сделать класс конкретным;
- Удаление
  - удалить класс;
  - удалить метод.

Обозначив методы рефакторинга и свойства архитектуры, необходимо также определить, как оценивать полученный результат. Составив функцию, зависимую от числового выражения вышеперечисленных свойств, можно получить

$$q_d = \sum_{m=1}^{n} w_m metric_m(d). \quad (1)$$

В формуле (1): q_d – значение качества полученной архитектуры; w_m – вес метрики m; $metric_m(d)$ – значение свойства m в архитектуре d. Отсюда следует вывод, что свойства или же метрики должны быть взвешенными. Веса были расставлены в таблице 1 в соответствии с описанием приоритетов того или иного свойства в работе [19].

Таблица 1. Описание весов метрик

Метрика	Количество абстрактных суперклассов	Количество повторяющихся методов	Количество неиспользуемых методов	Количество безликих классов
Вес	3	2	1	-1

Последним шагом являлся выбор того, каким образом будет осуществляться переход между вариантами архитектуры. Поскольку задача фактически сводится к задаче глобальной оптимизации, необходимо было выбрать алгоритмический метод решения задач соответствующего класса [20]. Выбор был сделан в пользу метода имитации отжига [21]. В целом для решения данной задачи можно было использовать и любой другой похожий алгоритм, однако обычные стохастические методы поиска в этом случае требовали бы крайне много времени для получения результата [22]. В добавление к этому отметим, что данный метод обладает более успешными возможностями выхода из локальных минимумов. В качестве недостатка можно отметить, что не будет возможности доказать оптимальность полученного решения. Однако, поскольку в данном случае достаточно будет

найти качественное решение за ограниченное время, а не самое лучшее, этого алгоритма будет вполне достаточно.

РЕЗУЛЬТАТЫ

Для проверки качества работы полученного решения была создана небольшая иерархия классов, представленная на рисунке 1. Заглавные буквы в названии класса означают, что данный класс использует указанные методы. Как можно заметить, исходная структура классов имеет следующие недостатки: классы ConcreteAB, ConcreteABC, ConcreteABDE, ConcreteABD наследуют метод f, но не используют его, а метод b дублируется в классах ConcreteABDE, ConcreteAB, ConcreteABC.

Реализовав данный алгоритм на языке Python и применив его на протяжении 1000 эпох к данной структуре, мы получили новую иерархию классов, представленную на рисунке 2. Как можно увидеть, вышеперечисленные проблемы были решены путем добавления новых абстрактных классов AbstractBaseSubClass и AbstractABDE и выносом соответствующих методов в них. Это позволило избежать дублирования методов и добиться того, что классы наследуют только те методы, которые используют.

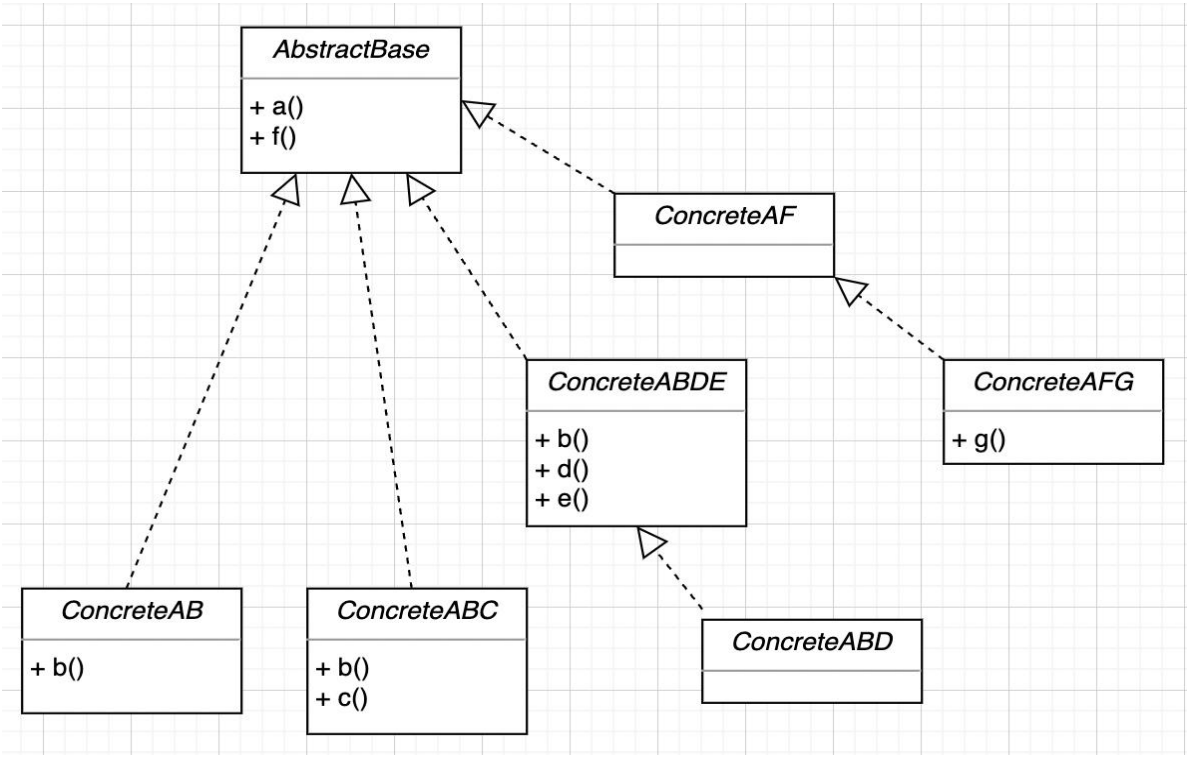

Рис. 1. Изначальная иерархия классов

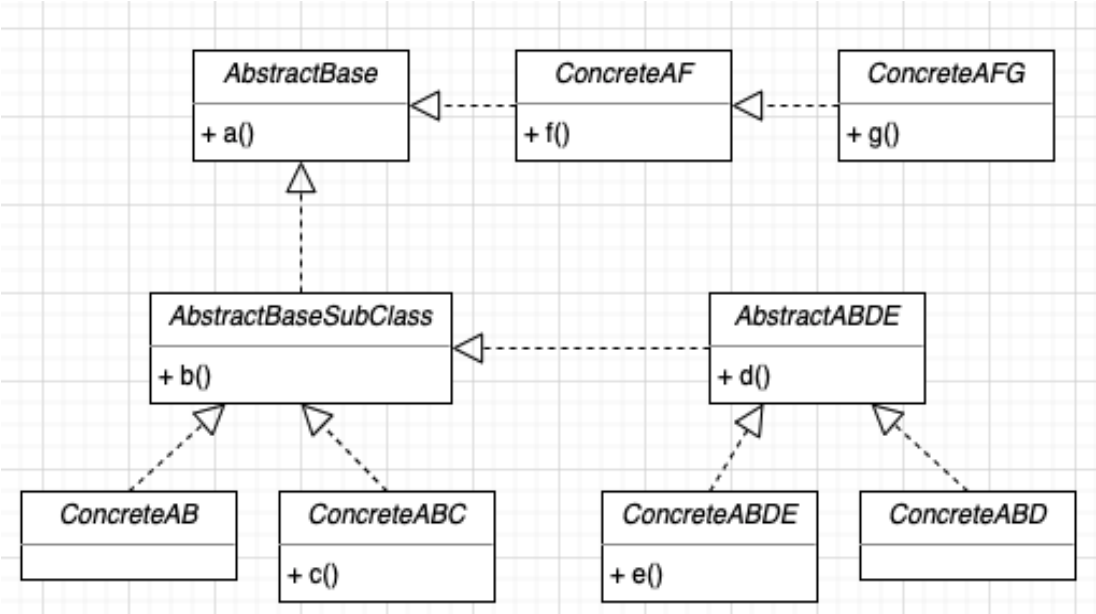

Рис. 2. Конечная иерархия классов

ОБСУЖДЕНИЕ

В результате исследования этапа проектирования ПО были проанализированы текущие способы его автоматизации, а также предложено собственное решение для автоматизации проектирования архитектуры ПО. Предложенное решение реализует автоматизацию рефакторинга структуры классов на основе метода комбинаторной оптимизации, что частично способствует автоматизации проектирования ПО и, как следствие, снижению стоимости этапа проектирования ПО. На основании выбранных метрик качества архитектуры и составленной целевой функции задача была сведена к задаче глобальной оптимизации. После этого на основе примера реальной структуры классов была протестирована работа алгоритма.

Набор метрик, выбранный в данной работе, можно заменить на любой другой, для этого достаточно будет определиться с их составом, расставить приоритеты, выбрав веса, и определить способ их подсчета в структуре классов.

Перспективой развития данной работы можно считать тестирование алгоритма на реальной достаточно крупной структуре классов и оценку его с точки зрения реальной практической пользы, а также подбор дополнительных метрик оценки качества архитектуры и методов ее изменения.

СПИСОК ЛИТЕРАТУРЫ

1. *Водзинская Э.В.* Оценка стоимости компаний российского рынка разработки программного обеспечения методами DCF и EVA // Экономические исследования и разработки. 2016. № 4. С. 163–168.

2. *Щенников А.Н.* Проектирование программного обеспечения для информационных систем. Saarbruken: LAP LAMBERT, 2018. 126 с.

3. *Макконнелл С.* Совершенный код. СПб.: Питер, 2005. 59 с.

4. *Фаулер М.* Архитектура корпоративных программных приложений: Пер. с англ. М.: Издательский дом Вильямс, 2006. 544 с.

5. *Влацкая И.В., Заельская Н.А., Надточий Н.С.* Проектирование и реализация прикладного программного обеспечения: учебное пособие. Оренбург: Оренбургский гос. ун-т, 2015. 118 с.

6. *Fox M.S., Gruninger M.* Enterprise modeling // AI magazine. 1998. Vol. 19. No. 3. 109 p. https://doi.org/10.1609/aimag.v19i3.1399

7. *Miksa K. et al.* Case Studies for Marrying Ontology and Software Technologies // Ontology-Driven Software Development. Springer, Berlin, Heidelberg, 2013. P. 69–94.

8. *Happel H.J. et al.* KOntoR: an ontology-enabled approach to software reuse // In: Proc. of The 18Th Int. Conf. On Software Engineering and Knowledge Engineering. 2006. P. 91.

9. *Borges Ruy F. et al.* SEON: A software engineering ontology network // European Knowledge Acquisition Workshop. Springer, Cham, 2016. P. 527–542.

10. *Chauvel F., Jézéquel J.M*. Code generation from UML models with semantic variation points // International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, 2005. P. 54–68.

11. *Маклаков С.В.* BPwin и ERwin. CASE-средства разработки информационных систем. М.: Диалог-мифи, 2001. 121 с.

12. *Lakin R., Capon N., Botten N.* BPR enabling software for the financial services industry // Management services. 1996. Vol. 40. No. 3. P. 18–20.

13. *Gryphon R.* Design better apps with SilverRun // Data Based Advisor. 1994. Vol. 12. No. 1. P. 103–107.

14. *Quatrani T.* Visual modeling with Rational Rose 2000 and UML. Addison-Wesley Professional, Second Edition. Addison Wesley, 2000. 288 p.

15. *Kopyltsov A.V. et al.* Algorithm of estimation and correction of wireless telecommunications quality // 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA). IEEE, 2018. P. 1–4.

16. *Vathsavayi S. et al.* Tool support for software architecture design with genetic algorithms // 2010 Fifth International Conference on Software Engineering Advances. IEEE, 2010. P. 359–366.

17. *Мейер Б.* Объектно-ориентированное программирование и программная инженерия: учебное пособие. 2-е изд., испр. М.: Национальный Открытый Университет «ИНТУИТ», 2016. 286 с.
URL: https://biblioclub.ru/index.php?page=book&id=429034

18. *Джамшиди П. и др.* Микросервисы: пройденный путь и дальнейшие цели // Открытые системы. СУБД. 2018. № 3. С. 19–23.

19. *Riel A.J.* Object-Oriented Design Heuristics. Addison-Wesley Professional; Illustrated edition, 1996. 400 p.

20. *Орлянская И.В.* Современные подходы к построению методов глобальной оптимизации // Исследовано в России. 2002. Т. 5. С. 2097–2108.

21. *Глушань В.М.* Метод имитации отжига // Известия Южного федерального университета. Технические науки. 2003. Т. 31. № 2. С. 148–150.

22. *Матренин П.В., Гриф М.Г., Секаев В.Г.* Методы стохастической оптимизации: учеб. пособие. Новосибирск: Изд-во НГТУ, 2016. 66 с.

DEVELOPMENT OF THE EXPERT SYSTEM FOR BUILDING THE ARCHITECTURE OF SOFTWARE PRODUCTS

Andrey Grishin[1] [0000-0002-7355-4878], Karen Grigoryan[2] [0000-0001-6470-1832]

[1, 2]Kazan (Volga Region) Federal University, 35 Kremlevskaya str., Kazan, 420008

[1]andrey.grishin.work@gmail.com, [2]karigri@yandex.ru

Abstract

The article is devoted to automation of the software design stage. In the course of the study, the reasons for the high importance of this stage and the relevance of its automation were analyzed. The main stages of this stage were also considered and the existing systems that allow automating each of them were considered. In addition, an own solution was proposed within the framework of the problem of class structure refactoring based on the combinatorial optimization method. A solution method has been developed to improve the quality of the class hierarchy and tested on a real model.

Keywords*: automation, design, refactoring, software architecture, OOP, optimization.*

REFERENCES

1. *Vodzinskaya E.V.* Ocenka stoimosti kompanij rossijskogo rynka razrabotki programmnogo obespecheniya metodami DCF i EVA // Ekonomicheskie issledovaniya i razrabotki. 2016. № 4. S. 163–168.
2. *Shchennikov A.N.* Proektirovanie programmnogo obespecheniya dlya informacionnyh sistem. Saarbruken: LAP LAMBERT, 2018. 126 s.
3. *Makkonnell S.* Sovershennyj kod. SPb.: Piter, 2005. 59 s.
4. *Fauler M.* Arhitektura korporativnyh programmnyh prilozhenij: Per. s angl. M.: Izdatel'skij dom Vil'yams, 2006. 544 s.
5. *Vlackaya I.V., Zael'skaya N.A., Nadtochij N.S.* Proektirovanie i realizaciya prikladnogo programmnogo obespecheniya: uchebnoe posobie. Orenburg: Orenburgskij gos. un-t., 2015. 118 s.

6. *Fox M.S., Gruninger M.* Enterprise modeling // AI magazine. 1998. Vol. 19. No. 3. 109 p. https://doi.org/10.1609/aimag.v19i3.1399.

7. *Miksa K. et al.* Case Studies for Marrying Ontology and Software Technologies // Ontology-Driven Software Development. Springer, Berlin, Heidelberg, 2013. P. 69–94.

8. *Happel H.J.* et al. KOntoR: an ontology-enabled approach to software reuse // In: Proc. of The 18Th Int. Conf. on Software Engineering and Knowledge Engineering. 2006. P. 91.

9. *Borges Ruy F. et al.* SEON: A software engineering ontology network // European Knowledge Acquisition Workshop. Springer, Cham, 2016. P. 527–542.

10. *Chauvel F., Jézéquel J.M.* Code generation from UML models with semantic variation points // International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, 2005. P. 54–68.

11. *Maklakov S.V.* BPwin i ERwin. CASE-sredstva razrabotki informacionnyh sistem. M.: Dialog-mifi, 2001. 121 s.

12. *Lakin R., Capon N., Botten N.* BPR enabling software for the financial services industry // Management services. 1996. Vol. 40. No. 3. P. 18–20.

13. *Gryphon R.* Design better apps with SilverRun // Data Based Advisor. 1994. Vol. 12. No. 1. P. 103–107.

14. *Quatrani T.* Visual modeling with Rational Rose 2000 and UML. Addison-Wesley Professional, Second Edition. Addison Wesley, 2000. 288 p.

15. *Kopyltsov A.V. et al.* Algorithm of estimation and correction of wireless telecommunications quality // 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA). IEEE, 2018. P. 1–4.

16. *Vathsavayi S. et al.* Tool support for software architecture design with genetic algorithms // 2010 Fifth International Conference on Software Engineering Advances. IEEE, 2010. P. 359–366.

17. *Mejer B.* Ob"ektno-orientirovannoe programmirovanie i programmnaya inzheneriya: uchebnoe posobie. 2-e izd., ispr. M.: Nacional'nyj Otkrytyj Universitet «INTUIT», 2016. 286 s. URL: https://biblioclub.ru/index.php?page=book&id=429034

18. *Dzhamshidi P. i dr.* Mikroservisy: projdennyj put' i dal'nejshie celi // Otkrytye sistemy. SUBD. 2018. № 3. S. 19–23.

19. *Riel A.J.* Object-Oriented Design Heuristics. Addison-Wesley Professional; Illustrated edition, 1996. 400 p.

20. *Orlyanskaya I.V.* Sovremennye podhody k postroeniyu metodov global'noj optimizacii // Issledovano v Rossii. 2002. T. 5. S. 2097–2108.

21. *Glushan' V.M.* Metod imitacii otzhiga // Izvestiya YUzhnogo federal'nogo universiteta. Tekhnicheskie nauki. 2003. T. 31. № 2. S. 148–150.

22. *Matrenin P.V., Grif M.G., Sekaev V.G.* Metody stohasticheskoj optimizacii: ucheb. posobie. Novosibirsk: Izd-vo NGTU, 2016. 66 s.

СВЕДЕНИЯ ОБ АВТОРАХ

ГРИШИН Андрей Евгеньевич – магистрант, Казанский (Приволжский) федеральный университет, г. Казань.

Andrey Evgenyevich GRISHIN – graduate, Kazan (Volga region) Federal University, Kazan.

Email: andrey.grishin.work@gmail.com
ORCID: 0000-0001-6470-1832

ГРИГОРЯН Карен Альбертович – кандидат экономических наук, доцент, Казанский (Приволжский) федеральный университет, г. Казань.

Karen Albertovich GRIGORIAN – Candidate of Economics, Associate Professor, Kazan (Volga region) Federal University, Kazan.

Email: karigri@yandex.ru
ORCID: 0000-0001-6470-1832

Материал поступил в редакцию 20 мая 2022 года