Математические структуры и моделирование
2011, вып. 22, с. 55–68

УДК 004.4'24

РАЗРАБОТКА СЕМЕЙСТВА ПРОГРАММНЫХ СИСТЕМ В СПЕЦИФИЧЕСКОЙ ПРЕДМЕТНОЙ ОБЛАСТИ

С. В. Гусс

Рассматриваются общие вопросы разработки семейства программных систем в рамках линейки программных продуктов на основе систематического подхода к повторному использованию производственных активов (программных компонентов, специализированных инструментов). Проводится обзор оснований, лежащих в корне данной практики. Обращается внимание на профессиональные качества разработчиков конечных продуктов и требуемый уровень знаний для их эффективной работы.

Введение

Согласно [13, 15, 19, 21], специально созданные или отобранные, адаптированные и настроенные на определённое семейство программных систем, средства разработки в сочетании с определённой профессиональной подготовкой исполнителей проекта способны увеличить производительность процесса создания компьютерных приложений и обеспечить высокое качество разрабатываемого продукта.

К таким средствам относятся модели, каркасы, шаблоны и предметно-ориентированные языки моделирования/программирования, способные связать составные части приложения в рамках общей архитектуры семейства программных систем [2, 5, 19]. В моделях отображаются общие очертания программных единиц. Каркасы определяют характерные для программных единиц повторноиспользуемые абстракции. Предметно-ориентированные языки делают процесс работы с этими абстракциями производительным, а шаблоны повышают эффективность процесса, ограничивая возможность совершения некорректных действий, в то же время указывая верный путь работы с абстракциями. Добавив к этому автоматические программные инструкции, оказывающие помощь разработчикам, непосредственно во время создания приложений, можно объединить всё вместе в специальный пакет, расширяющий возможности интегрированной среды разработки. К примеру, среда Microsoft Visual Studio 2010

Омский государственный университет им. Ф. М. Достоевского.
E-mail: InfoGuss@Gmail.com

имеет в своём арсенале все необходимые для расширения средства, включая специальные шаблоны проектов, есть нечто подобное и в средах IDE Eclipse и Embarcadero RAD Studio. Особое выделение продуктов компании Microsoft вызвано тем, что в рамках одной из её исследовательских инициатив проводятся поиск и развитие методов автоматизации, а в конечном счёте и индустриализации разработки программных продуктов, что должно повысить профессиональный уровень зрелости разработчиков программного обеспечения. Эта инициатива, названная фабриками программного обеспечения [19], рассматривается автором статьи в своём теоретическом и практическом аспекте.

Наряду с общими рекомендациями по составлению пакета вспомогательных средств, куда входят библиотеки программных компонентов, специализированные инструменты, документация и автоматические руководства, описанными в [19, 21, 25], необходимы также конкретные рекомендации по созданию пакетов для конкретной предметной области [15]. В качестве предметной области в представленном в статье проекте программного обеспечения выступают игровые обучающие программные системы с лингвистической составляющей [6–8]. Такие программные системы оптимально подходят, с точки зрения автора (подкреплённой исследованиями учёных в области педагогики и психологии [9]), для учебного процесса.

О явлении компьютерных игр в целом можно узнать из исследований в области культурологии, представленных в работе [14]. Вопрос влияния игр на психологию играющих и обучающихся с помощью них с разных сторон рассмотрен в [9]. Что же касается внутренней (не только технической) составляющей компьютерных игр, то довольно полно информация по данному вопросу представлена в [18].

О необходимости и практичности конкретизации современных методов разработки компьютерных приложений для отдельных предметных областей говорится во множестве работ, часть из которых будет упомянута и коротко рассмотрена в тексте статьи.

Кроме того, также рассматриваются и этические вопросы, касающиеся профессиональной разработки приложений для специальной предметной области. Автор упомянет и об исторических моментах, а также основах, лежащих в затронутых темах. Не упущен из виду и такой аспект профессиональной разработки, как человеческий фактор [13], связанный с психологическими особенностями разработчиков, их профессиональными качествами и набором необходимых им знаний [17]. Данные вопросы рассматриваются в общих чертах, они основываются на существующих сегодня знаниях, тем не менее для тех, кто хочет узнать об этом подробней, приводятся литературные источники.

1. Профессиональная разработка программного обеспечения

В наше время от специалистов в области программной инженерии, непосредственно от профессиональных разработчиков требуются определённые знания о производстве программных систем «в определённой тематической области их

применения» [13]. В [15] говорится, что для эффективного внедрения инноваций инженерии программного обеспечения необходимо выразить их в терминах, «понятных для каждой конкретной подотрасли». Это подразумевает необходимость конкретизации знаний в области разработки приложений для конкретной прикладной области [34]. Согласно [17], от сегодняшних разработчиков требуется больше, чем просто написание программного кода. Необходимо иметь представления обо всём жизненном цикле программного обеспечения [4,11] и в соответствии со своей специализацией уметь пользоваться специальными знаниями. Что самое главное, там также упоминается о том, что «эффективное использование специфических для предметной области методов, средств и компонент в большинстве случаев обеспечивает успешность разработок с использованием программной инженерии». Таким образом, аналитики, архитекторы, проектировщики и разработчики помимо знаний из инженерии программного обеспечения должны обладать хотя бы минимальной информацией из специальной предметной области [13]. Минимум информации в данном случае должен определяться уровнем «критичности» проекта программного обеспечения, т. е. тем, какое воздействие реализация данного проекта может оказать на своё окружение (в первую очередь исполнителей проекта и пользователей, а впоследствии и на общество в целом или его части). Именно здесь решающее значение имеет человеческий фактор. Значительный объём информации по этому вопросу можно найти в работе [13].

Всё это говорит в пользу подхода к профессиональной ***разработке в определённой предметной области***. Здесь применяется моделирование предметной области [1], появляются специальные термины и образуются некоторые закономерности, которые необходимо учитывать, а в определённых случаях требуется повышенная внимательность и способность длительной концентрации внимания [13]. В этих случаях даже оправдана повышенная предусмотрительность и скрупулёзность исполнителей, ответственных за критические аспекты проекта.

На основании [13] профессиональная разработка включает в себя аспекты ***качества***, ***управления*** и ***экономики***. Вопросы качества и управления имеют существенное значение для проекта, а вопросу экономики, особенно в случае больших проектов, уделяется особое внимание, в частности, согласно [12], формируется новая научная дисциплина — «экономика жизненного цикла программных средств». В рамках аспекта качества должны рассматриваться такие вопросы, имеющие отношение к нефункциональным требованиям программного обеспечения, как удобство, надёжность, производительность и поддерживаемость (сопровождаемость) [19]. Как видно, нефункциональные требования должны распространяться не только на конечный продукт, но и на процесс его производства.

2. Инструменты разработчика программных продуктов

В наборе сегодняшнего разработчика можно встретить такие инструменты, как модели [1, 23, 34], шаблоны [2, 16], предметно-ориентированные языки

[19, 21, 22, 32], компоненты [26, 28, 29, 33, 35], рекомендации и скрипты [19, 25], для автоматизации определённых частей процесса разработки программного обеспечения.

Все эти инструменты стали плодом развития дисциплины разработки компьютерных приложений. С развитием появлялись новые методы, новые инструменты, какие-то отбрасывались, а какие-то укоренялись и совершенствовались. Нельзя сказать, что до появления объектно-ориентированной парадигмы не было представленных выше инструментов, но массово о них заговорили именно с той поры, начиная с конца 1980-х гг. К этому времени уже была определённая теоретическая база, а развитие технологий позволило опробовать это всё на практике, внести коррективы и сделать новые предложения.

Одной из фундаментальных работ считается [26], представленная Малькольмом Дугласом Макилроем (Malcolm Douglas McIlroy) в конце 1960-х гг. на конференции, впервые посвящённой инженерии программного обеспечения. В этой работе делаются предположения о том, как и в какой форме могли бы быть полезны ***повторно используемые активы разработки*** программного обеспечения. В то время под активами понимались библиотеки с набором необходимого функционала. Конкретно предлагались методы каталогизации библиотек компонентов, сегодняшний же подход предполагает взамен метод генерации или автоматического подгона из общего представления [21]. Немалое развитие в этом направлении произвёл Дэвид Парнас (David Parnas), являющийся основоположником множества фундаментальных принципов разработки программного обеспечения. Его главные работы были посвящены процессу проектирования компонентов в рамках семейства родственных программных систем.

Что касается моделей, то большое увлечение ими началось с работ по созданию универсального языка моделирования, с помощью которого, используя единую систему обозначений для выражения своих намерений, могли бы взаимодействовать и профессионально общаться разработчики программных систем разных специализаций, от аналитика до программиста, реализующего отдельный модуль системы. Это вылилось в создание языка UML, и здесь не обойтись без упоминания Гради Буча (Grady Booch), Джеймса Рамбо (James Rumbaugh) и Айвара Джекобсона (Ivar Jacobson), объединивших и развивших всевозможные, в том числе и собственноручно разработанные, подходы к моделированию предметной области программного обеспечения [1].

Фундаментальный труд по шаблонам — работа [16], появившаяся в середине 1990-х гг. В рамках этого проекта по каталогизации шаблонов проектирования принимали участие Эрих Гамма (Erich Gamma), Ричард Хелм (Richard Helm), Ральф Джонсон (Ralph Johnson), Джон Влиссидес (John Vlissides). Однако вдохновителем идеи является Кристофер Александер (Christopher Alexander), не имеющий прямого отношения к компьютерным наукам, один из первых, кто заговорил о языке шаблонов, правда, применительно к архитектурным сооружениям. По некоторым сведениям, например [21], он не был удовлетворён своей теорией, обосновывая это тем, что архитектурные сооружения, построенные с использованием шаблонов, получаются слишком однообразными, что, к сча-

стью, не является проблемой для внутреннего строения проекта программного обеспечения, а, наоборот, идёт ему на пользу.

Предметно-ориентированные языки не являются чем-то очень новым, они были раньше и широко используются сегодня [32], когда платформенные технологии развиты настолько, что дают возможность создавать языки уже на имеющейся технологической базе. Это позволяет обойтись без создания компилятора и не требует от разработчиков детальных знаний процесса их создания. Имеются труды, из которых можно узнать об этом явлении (в частности, [19,22]) и даже ощутить его на практике (например, благодаря [22,32]). В работе [32] всё внимание уделяется разработке текстовых языков, а в работе [22] — графических, где описание сущностей происходит с помощью графических пиктограмм и линий с нанесением поясняющей информации. Первые хороши для решения крупных проблем. Вторые подходят для решения средних и небольших по масштабу задач, но требующих от разработчика определённой внимательности.

В рамках конкретной парадигмы разработки могут использоваться различные ***модели***, статические и динамические, описывающие структуру системы или отдельной её части и взаимодействие элементов внутри неё или с внешним миром. Для эффективного повторного использования основной проблемой, препятствующей повторному использованию модели, являются её границы. Границы мешают использовать её в другом контексте без добавления к ней определённых изменений [34]. Эта проблема частично решается с применением шаблонов проектирования.

Цель ***шаблонов*** — сохранить обобщённые знания экспертов с тем, чтобы впоследствии их можно было использовать в определённом контексте. В основном знания обобщаются в определённой ***горизонтальной предметной области*** (выделенной на основе классов частей систем, исходя из функциональных возможностей) с тем, чтобы их можно было использовать в контексте ***вертикальной предметной области*** (выделенной на основе классов систем согласно области применения). Чтобы эффективно применять шаблоны, в частности проектирования, нужно, во-первых, иметь о них представление, а во-вторых, желательно иметь опыт их применения, которого может и не быть или он утратился со временем. Одно из решений проблемы — встроить эти знания в каркас.

Каркас облегчает разработку системы в целом или отдельной её части, благодаря уже имеющейся в наличии общей структуры будущей программной единицы. От разработчика требуется лишь произвести детализацию каркаса. Возможная проблема состоит в том, что прежде чем использовать каркас, нужно знать, как производить его настройку: если каркас большой, ситуация усложняется. Для облегчения работы с каркасом можно использовать специально созданные для него предметно-ориентированные языки.

Главное преимущество ***предметно-ориентированных языков*** — облегчение процесса детализации каркаса и возможность с их стороны, благодаря встроенным программным рекомендациям и скриптам, объединять компоненты или составляющие в рамках компонента без рутинной работы со стороны разработчика. Одни языки могут применяться для решения мелких проблем в рамках

отдельного компонента, другие — для решения крупных проблем в рамках целой системы. Видно, что перечисленные инструменты взаимосвязаны, и, чтобы избавить разработчиков от путаницы в их использовании и неудобств, связанных с совместимостью, целесообразно объединить их в рамках единого пакета. Подробнее этот вопрос будет раскрыт в разделе, посвящённом фабрике разработки программ.

Стоит сказать ещё несколько слов о подходах, которые оказали непосредственное влияние на появление этих инструментов. В [19, 21, 22] представлены следующие подходы, имеющие отношение к разработке программного обеспечения, учитывая особенности определённой предметной области.

Разработка с использованием моделей (Model-Driven Development). В основе подхода лежит идея, что из модели можно сгенерировать пригодный для применения код. Одно из перспективных направлений подхода — ***архитектура, управляемая моделью*** (Model-Driven Architecture) [27], с которым связаны такие понятия, как «машинно-независимая» и «платформо-независимая модель». ***Языкоориентированное программирование*** (Language-Oriented Programming). Предполагает создание специального языка для решения определённых задач программирования. ***Предметно-ориентированное моделирование*** (Domain-Specific Modeling). Объединяет ряд направлений, связанных идеей создания предметно-ориентированных языков для решения задач определённой предметной области. ***Родовое программирование*** (Generic Programming). Направлено на решение проблемы структурирования компонентов реализации, из которых строятся конечные программные системы. Связано с поиском наиболее абстрактных представлений эффективных решений для оптимального повторного использования компонентов. ***Ментальное программирование*** (Intentional Programming). Главная идея, являющаяся трамплином для двух далее перечисленных подходов, такова — необходима расширяемая среда программирования и метапрограммирования. Предполагает автоматизированное редактирование и рефакторинг кода. Предусматривает решение проблем спутывания кода, включая тем самым идеи ***аспектно-ориентированного программирования*** (Aspect-Oriented Programming). ***Порождающее программирование*** (Generative Programming). Под идеей автоматической (в идеале) генерации приложений объединяет такие темы, как инженерия предметной области, моделирование характеристик и различные техники генерации программ. ***Фабрики разработки программ*** (Software Factories). Подход включает в общем виде идею, представленную в данной статье, а именно: объединение статического содержимого (шаблонов, моделей, предметно-ориентированных языков, компонентов и инструкций) с динамическим содержимым (поддающимися настройке процессами, мастерами, тестами) в рамках пакета, расширяющего интегрированную среду разработки. Несмотря на то, что два последних подхода предлагают генерацию приложений или его частей из моделей, они, тем не менее, предусматривают работу со стороны разработчика, связанную с добавлением программного кода. Это — основное отличие от ***CASE-технологий*** и ***архитектуры, управляемой моделью***, которые предлагают генерацию предложения «нажатием одной кнопки».

3. Инженерия предметной области

Существует множество работ, излагающих идеи инженерии предметной области. Здесь можно выделить два класса направлений. Первый класс составляют работы, описывающие идеи для создания методов разработки приложений, второй класс составляют работы, предлагающие непосредственно методы разработки, на основе идей первого класса и их практическое применение. В качестве примера первого класса можно привести [34], в качестве примера второго класса — [10]. В работе [34] представлена так называемая теория предметной области (Domain theory) для разработки программного обеспечения, а в работе [10] — применение приёмов различных парадигм разработки, в том числе связанных с теорией предметной области, на практике. Используется язык программирования C++.

В работе [21] проведён обзор различных методов анализа и инженерии предметной области. Наибольшее внимание в ней уделяется методу ***характеристико-ориентированного анализа предметной области*** (Feature-Oriented Domain Analysis) и ***методу моделирования предметной области организации*** (Organization Domain Modeling).

Инженерия предметной области (Domain Engineering) имеет прямое отношение к вопросу повторного использования. Согласно определению, взятому из [21], она имеет дело со сбором, систематизацией и сохранением накопленного опыта создания программных систем в «форме средств многократного применения в рамках определённой предметной области». Кроме того, она обеспечивает методы, способствующие повторному использованию этих средств, в процессе разработки новых приложений, облегчает их поиск, классификацию, распространение, адаптацию и сборку.

Главное отличие от традиционных методов разработки состоит в следующем. Во-первых, на стадии анализа, вместо определения требований к отдельной системе, проводится определение требований к семейству систем. Во-вторых, на стадии проектирования вместо разработки проекта отдельной системы проводится разработка проекта семейства систем, ну а на этапе реализации, помимо реализации самой системы, производятся многократно используемые компоненты, которые можно будет использовать в будущем.

Тема повторного использования применительно к разработке программных продуктов весьма полно представлена в работе [33]. Примечательно, что в работе, наряду с организационными аспектами внедрения, представлены ***метрики повторного использования***. К ним относятся атрибутные, структурные, функциональные метрики, метрики, характерные для всего приложения в целом, метрики инженерии предметной области и метрики организационного уровня. Практика введения систематического повторного использования главным образом предполагает создание проблемно и предметно ориентированных абстракций, а также каркасов, в рамках которых эти повторно-используемые абстракции могут быть разработаны путём детализации. Современная тенденция предполагает наличие предметно-ориентированного языка для детализации каркаса, тем самым решается множество проблем, связанных с человеческим

фактором, таких как уровень знаний и объём единовременного восприятия информации в процессе разработки компонента.

Систематическое повторное использование указывает на необходимость разработки не отдельного приложения, а семейства программных систем. А это в свою очередь решается введением в производственный процесс такой технологии, как линейки программных продуктов. С разработкой семейства связаны такие вопросы, как общность и изменчивость характеристик членов семейства. Существуют различные подходы для выявления этих особенностей.

Характерной особенностью ***семейства программных систем*** является наличие обобщённой архитектуры, описывающей всё множество членов конкретного семейства, указывая при этом так называемые точки изменчивости, т. е. те места, в которых имеют место различия между членами семейства.

Линейки программных продуктов реализуют семейство программных систем, применяя имеющиеся производственные активы в рамках организационного процесса. К производственным активам относятся архитектурные образцы, шаблоны проектирования, языки, каркасы, компоненты и т. д. Организационный процесс накладывает определённые ограничения на продукт и его производство. К ограничениям на продукт относятся различные требования к стандартам, интерфейсам, пакетированию, локализации, лицензированию. К ограничениям на производство относятся требования к выбору технологий, платформ, критериям приёмки и отчётности, сюда также относятся требования к бюджету и планированию.

Основополагающей работой в области разработки семейства программных систем является [30]. Заслуживающие внимания работы, касающиеся линеек программных продуктов, — [24, 31].

4. Фабрики разработки программ

Фундаментальной работой по фабрикам разработки программ является работа [19] авторского коллектива во главе с Джеком Гринфилдом (Jack Greenfield). Практическому аспекту фабрик программного обеспечения посвящена работа [25] Гунтера Ленца (Gunther Lenz).

Фабрика разработки программного обеспечения представляет собой линейку программных продуктов, которая благодаря шаблону, основанному на схеме, называемой схемой фабрики программного обеспечения, позволяет конфигурировать инструменты и процессы для автоматизации разработки. Фабрика предполагает наличие специальных компонентов, способных к адаптации, сборке и конфигурированию на основе каркаса.

Основные идеи фабрик программного обеспечения — повышение уровня индустриализации отрасли разработки программного обеспечения, повышение экономической эффективности последующих приложений из определённого класса, экономия за счёт области применения (благодаря уже решённым подпроблемам предметной области), систематическое повторное использование.

Построение семейства приложений происходит в несколько этапов.

Первый этап — ***разработка линейки продуктов***. Включает в себя три подэтапа, состоящих из отдельных фаз.

Первый подэтап — ***анализ линейки продуктов***. Здесь решается вопрос относительно того, какие продукты будут производиться фабрикой. Первая фаза — ***определение линейки продуктов***. Цель — достичь понимания того, как архитектура, учитывающая изменчивость, будет способствовать построению множества схожих, но в то же время уникальных программных продуктов. Также необходимо идентифицировать и зафиксировать проблемы, в решении которых заинтересованы пользователи, и выявить продукты, которые будут способны решить эти проблемы. Немаловажное значение имеет описание контекста, в котором будут использоваться продукты. Вторая фаза — ***определение границ предметной области***, третья фаза — ***определение границ решений***. На второй фазе рассматривается множество проблем, на третьей — множество решений, т. е. то, как именно будут решаться проблемы, описанные во второй фазе.

Второй подэтап — ***дизайн линейки продуктов***. Здесь решается главный вопрос — как именно фабрика будет производить продукты. Первая фаза — ***разработка архитектуры***. Производится поиск необходимых архитектурных шаблонов. Здесь же рассматриваются вопросы о возможности создания специальных инструментов, таких как каркас и предметно-ориентированный язык для использования абстракций, представленных каркасом. Вторая фаза — ***отображение требований***. На этой фазе производится отображение изменчивости требований на изменчивость производственных активов, анализируется зависимость между изменчивыми элементами. Третья фаза — ***определение и автоматизация процесса***.

Третий подэтап — ***реализация линейки продуктов***. Первая фаза — ***обеспечение активами***. На данной фазе строятся или приобретаются необходимые активы. Вторая фаза — ***пакетирование активов***. Здесь все производственные активы упаковываются в шаблон фабрики программного обеспечения.

Итак, на этапе разработки линейки продуктов получен ***шаблон фабрики программного обеспечения***. Будучи установленным в интегрированную среду разработки, данный шаблон становится ***фабрикой разработки программ***.

Следующий этап — ***разработка продуктов***. В роли разработчиков может выступать компания, разработавшая фабрику программного обеспечения. С тем же успехом она может быть использована и сторонними разработчиками, которые приобрели данную фабрику. Разработка продукта включает ***спецификацию продукта***, когда требования выражаются в терминах линейки продуктов с отображением их на производственные активы. То есть внимание уделяется не всему приложению, как при традиционном способе разработки, а отдельным частям уже готового приложения.

5. Фабрика разработки обучающих игровых программных систем

В этом разделе приводятся ответы на главные вопросы, которые стоят в последнем проекте автора, связанном с разработкой фабрики программного обеспечения.

Вопрос № 1. Кто будет использовать фабрику, а кто потреблять продукты, выработанные ею?

Ответ. Фабрику разработки обучающих игровых программных систем лингвистической направленности будут использовать небольшие компании-разработчики программного обеспечения, нацеленные на рынок электронного обучения, решившие занять на этом рынке определённое положение. Это также могут быть специалисты в сфере образования, обладающие знаниями разработки компьютерных приложений и желающие создать себе в помощь продукт, способный решать определённые педагогические задачи, встающие в процессе обучения ими своих подопечных. Продукты, выработанные фабрикой, будут использоваться в учебных заведениях и местах, где есть люди, желающие пройти обучение. Основная задача продукта — способность поддерживать процесс обучения таким образом, чтобы обучающийся не ощущал разницы между обучением, будь он в специальном учебном заведении или дома. То есть необходима эффективная система контроля и помощи со стороны программной системы.

Целесообразность использования фабрики заключается в том, что она позволяет строить приложения, отвечающие требованиям множества заказчиков. Нет необходимости строить излишне перегруженную функциями программную систему обучения, которая может лишь отпугнуть потенциальных заказчиков. А вот лёгкое в освоении приложение с минимальной функциональностью, но выполняющее именно то, что необходимо конкретному заказчику, как раз то, что нужно. Экономия средств в данном случае достигается за счёт применения повторно-используемых элементов для разработки приложений разным заказчикам со своим набором требований, большая часть которых совпадает, а вариации невелики, но обязательны для учёта.

Вопрос № 2. Какие необходимы знания разработчикам, использующим фабрику?

Ответ. Во-первых, это знание процессов, протекающих в ходе учебного процесса, а также — в какой степени продукты фабрики могут их поддерживать. Это нужно для того, чтобы была возможность оценить объём работы по созданию элементов поддержки процессов, не предусмотренных фабрикой. Во-вторых, необходимы знания того, как можно реализовать требования заказчиков, понимать, какие инструменты разработки для этого доступны.

Определить возможности продукта можно, ознакомившись со схемой фабрики программного обеспечения. Схема — это прежде всего способ упорядочивания элементов, использующихся в процессе разработки, иначе говоря, артефактов разработки, таких как модели, файлы исходного кода, манифесты развёртывания и т. д. Говоря образно, их упорядочивание происходит в сет-

ке, где каждый элемент, или ячейка, представляет собой определённую точку зрения, или аспект программного обеспечения. Вначале определяются инструменты для построения продуктов, т. е. языки, каркасы, шаблоны, а уже затем каждой ячейке сопоставляются реальные, пригодные для использования артефакты разработки. Один и тот же инструмент разработки может быть сопоставлен с несколькими ячейками. То есть, например, для работы с несколькими аспектами можно использовать один предметно-ориентированный язык. В конечном счёте сетка преобразуется в схему, где между ячейками уже существуют определённые связи и наложены ограничения. Схема — это также шаблон для описания продуктов фабрики. В [19] приводится удачная аналогия, схема — рецепт, точки зрения — ингредиенты (а также инструменты и процессы приготовления).

Вопрос № 3. Каковы существенные составляющие схемы фабрики в общих чертах?

Ответ. ***Первая категория*** составляющих — требования. Список функциональных требований весьма широк. Делится на категории систем, из которых состоит продукт обучения, получаемый с помощью фабрики. Итак, есть требования к функциональности следующих систем: управления пользователями, обучения, обслуживания, взаимодействия человек-машина, безопасности, коммуникации, учебного материала. В рамках перечисленных систем происходят определённые процессы, которые можно конфигурировать. Нефункциональные требования представлены следующими требованиями. Первая группа — требования к удобству, решают вопросы того, насколько быстро обучаемые могут освоить систему, эффективно ли организована система взаимодействия с системой, не препятствуют ли определённые элементы организации интерфейса продуктивной работе и т. д. Вторая группа — требования к надёжности систем. Решают, какие возможные дефекты системы являются терпимыми, а какие крайне не желательными, препятствующими продуктивному процессу обучения. Содержит требования к нормальному функционированию, защите от сбоев и безопасности. Третья группа — требования к производительности, учитывающие время выполнения определённых операций, пропускную способность и эффективность потребления ресурсов вычислительной машины. Четвёртая группа — требования к сопровождаемости: они включают в себя требования к лёгкости исправления дефектов, модификации, замены, переносимости на другие платформы. ***Вторая категория*** — общая архитектура продуктов и инфраструктура развёртывания. Отвечает на следующие вопросы. Из каких элементов состоит, какой стиль выбран, какие шаблоны предполагает. Как производится настройка продукта? Последний вопрос связан с таким аспектом, как особенность конкретной платформы, которая накладывает определённые ограничения на развёртывание продуктов. ***Третья категория*** — исполняемые артефакты разработки. Сюда входят компоненты, каркасы, конфигурационные файлы, отдельные классы и т. д., отсортированные по принадлежности к конкретной подсистеме (обучения, обслуживания и т. д.). Подробнее о составляющих подсистем можно узнать из работы [8].

Между составляющими схемы могут существовать различные отношения, которые необходимо учитывать и поддерживать их согласованность, чтобы избежать проблем, связанных с перекрытием информации между артефактами и их переименованием. Для разработки конкретного продукта необходимо сконфигурировать схему, а для этого схема должна содержать, помимо фиксированных частей, части изменяемые. Схема фабрики — это простое описание активов, используемых для построения. Реализацией схемы является шаблон фабрики программного обеспечения.

Вопрос № 4. Из каких инструментов состоит шаблон фабрики?

Ответ. Далее перечисляются инструменты и их описание.

Предметно-ориентированные языки. В рамках разрабатываемой фабрики их несколько для каждой из семи подсистем, решающих определённые аспекты. К примеру, язык для установления отношений между составляющими подсистем. Предполагается создание графических и текстовых языков.

Каркасы. Для каждой системы в фабрике представлен свой каркас, содержащий высокоуровневые абстрактные сущности, поддающиеся детализации и переопределению.

Шаблоны проектирования. В большинстве своём это поддержка классических шаблонов, представленных в [16].

Упакованные вместе, эти активы могут использоваться для разработки продуктов в интегрированной среде разработки. Шаблон, так же как и схему, необходимо сконфигурировать перед использованием. О побуждающих мотивах построения фабрики обучающих игровых программных систем лингвистической направленности и предшествующих этому проектах можно узнать из работы [7].

Заключение

В статье представлена информация о процессе разработки программного обеспечения, сфокусированном на систематическом повторном использовании производственных средств и инструментов для их адаптации под конкретный продукт в рамках семейства программных систем.

К сожалению, в рамках статьи невозможно полностью и в деталях описать все аспекты процесса создания пакета повторно используемых элементов и инструментов. Тем не менее автор полагает, что информации в данной статье достаточно для того, чтобы приступить к ознакомлению с подходами, способствующими поднятию зрелости индустрии разработки программного обеспечения. А представленный список литературы и указание на источники в соответствующих разделах поможет получить интересующимся необходимые знания по определённой теме. Для тех, кто желает подробнее ознакомиться именно с инженерной составляющей процесса разработки программного обеспечения, рекомендуется начать с классических работ Дэвида Парнаса (David Parnas) [28, 30] и Барри Боэма (Barry Boehm) [3, 20].

Литература

1. Арлоу Д., Нейштадт А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. 2-е изд. СПб.: Символ-Плюс, 2007. 624 с.
2. Басс Л., Клементс П., Кацман Р. Архитектура программного обеспечения на практике. 2-е изд. СПб.: Питер, 2006. 575 с.
3. Боэм Б. У. Инженерное проектирование программного обеспечения. М.: Радио и связь, 1985. 512 с.
4. Брауде Э. Технология разработки программного обеспечения. СПб.: Питер, 2004. 655 с.
5. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. СПб.: Символ-Плюс, 2001. 304 с.
6. Гусс С. В. Модель каркаса программных компонентов поддержки занятий лингвистической направленности в игровой форме // Прикладная информатика. 2010. Вып. 3 (27). С. 62–77.
7. Гусс С. В. Фабрика разработки игровых лингвистических программных систем учебного назначения // Технологии Microsoft в теории и практике программирования: материалы конференции / под ред. проф. В. П. Гергеля. Нижний Новгород: Изд-во Нижегород. гос. ун-та, 2010. С. 110–112.
8. Гусс С. В. Элементы повторного использования для программных систем учебного назначения. Концептуальное проектирование и анализ // Математические структуры и моделирование. 2009. Вып. 20. С. 78–92.
9. Керделлан К., Грезийон Г. Дети процессора: Как интернет и видеоигры формируют завтрашних взрослых. Екатеринбург: У-Фактория, 2006. 272 с.
10. Коплиен Дж. Мультипарадигменное проектирование для C++. СПб.: Питер, 2005. 235 с.
11. Липаев В. В. Программная инженерия. Методологические основы. М.: ТЕИС, 2006. 608 с.
12. Липаев В. В. Технико-экономическое обоснование проектов сложных программных средств. М.: СИНТЕГ, 2004. 284 с.
13. Липаев В. В. Человеческие факторы в программной инженерии: рекомендации и требования к профессиональной квалификации специалистов. М.: СИНТЕГ, 2009. 328 с.
14. Липков А. И. Ящик Пандоры: Феномен компьютерных игр в мире и в России. М.: Изд-во ЛКИ, 2008. 192 с.
15. Макконнелл С. Профессиональная разработка программного обеспечения. СПб.: Символ-Плюс, 2006. 240 с.
16. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. СПб.: Питер, 2008. 366 с.
17. Рекомендации по преподаванию программной инженерии и информатики в университетах = Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering; Computing Curricula 2001: Computer Science: пер. с англ. М.: ИНТУИТ.РУ, 2007. 462 с.
18. Роллингз Э., Моррис Д. Проектирование и архитектура игр: пер. с англ. М.: Вильямс, 2006. 1040 с.

19. Фабрики разработки программ: потоковая сборка типовых приложений, моделирование, структуры и инструменты / Д. Гринфилд [и др.]. М.: Вильямс, 2007. 592 с.
20. Характеристики качества программного обеспечения / Б. Боэм [и др.]. М.: Мир, 1981. 208 с.
21. Чарнецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. Для профессионалов. СПб.: Питер, 2005. 731 с.
22. Domain-Specific Development with Visual Studio DSL Tools / S. Cook [etc.]. USA: Addison Wesley Professional, 2007. 563 p.
23. Duffy D. Domain Architectures: Models and Architectures for UML Applications. USA: Wiley, 2004. 406 p.
24. Kang K., Sugumaran V., Park S. Applied Software Product Line Engineering. USA: CRC Press, 2010. 563 p.
25. Lenz G., Wienands C. Practical software factories in .NET. USA: Apress, 2006. 240 p.
26. McIlroy M. Mass Produced Software Components // Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, Scientific Affairs Division, NATO, Brussels, 1969. URL: http://cs.dartmouth.edu/~doug/components.txt (дата обращения: 28.09.2010).
27. MDA Distilled: Principles of Model-Driven Architecture / S. Mellor [etc.]. USA: Addison Wesley, 2004. 176 p.
28. Parnas D. L. Active Design Reviews: Principles and Practices // Proceedings of the 8th International Conference on Software Engineering, London, August 1985. URL: http://gala.cs.iastate.edu/coms309/references/ActiveDesignReviews.pdf (дата обращения: 28.09.2010).
29. Parnas D. L. On the Criteria to be Used in Decomposing Systems into Modules // Communications of the ACM, 15, 12, December 1972. URL: http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf (дата обращения: 28.09.2010).
30. Parnas D. L. On the Design and Development of Program Families // IEEE Transactions on Software Engineering. 1976. Vol. SE-2, No. 1. URL: http://web.cecs.pdx.edu/~omse532/Parnas_Families.pdf (дата обращения: 28.09.2010).
31. Parnas D. L. Software Product Lines: What to do when Enumeration won't Work // Proceedings of the First International Workshop on Variability Modelling of Software-Intensive Systems, January 16–18, 2007. URL: http://ulir.ul.ie/bitstream/10344/160/2/09SQRL1045.pdf (дата обращения: 28.09.2010).
32. Rahien A. DSLs in Boo. Domain-Specific Languages in .NET. USA: Manning Publications, 2010. 352 p.
33. Reuse-Based Software Engineering. Techniques, Organization, and Controls / H. Mili [etc.]. USA: John Wiley and Sons, 2002. 682 p.
34. Sutcliffe A. The Domain Theory: Patterns for Knowledge and Software Reuse. USA: CRC Press, 2002. 424 p.
35. Wang A., Qian K. Component-Oriented Programming. USA: Wiley-Interscience, 2005. 336 p.