ПРОГРАММНО-
ТЕХНИЧЕСКИЕ
КОМПЛЕКСЫ

Л.П. БАБЕНКО

УДК 681.3.06

ОНТОЛОГИЧЕСКИЙ ПОДХОД К СПЕЦИФИКАЦИИ СВОЙСТВ ПРОГРАММНЫХ СИСТЕМ И ИХ КОМПОНЕНТОВ

Ключевые слова: *готовые ресурсы разработки, управление знаниями, управление изменениями, модель знаний, модель проблемной области, онтология, повторное использование.*

Спектр продуктивных идей, наработанных в программной инженерии, в значительной мере связан с разными формами повторного использования, например, с базирующейся на компонентах разработкой программных систем, аспектным программированием, генераторами и компиляторами с универсальными или проблемно-ориентированными исходными языками и т.п. Главная идея повторного использования — накопление опыта, полученного при предшествующих разработках, в виде, удобном для построения будущих программных систем. Для этого необходимо соблюдение трех условий: материализованный опыт, доступная и понятная информация о нем и четкие правила его интеграции в новые разработки.

Будем называть готовыми ресурсами разработки (ГОР) представление опыта в виде коллекции артефактов, обеспечивающих решение определенной проблемы, общей для ряда программных разработкок, в заданном контексте. Контекст относится к одной из стадий жизненного цикла разработки (ЖЦ) программного продукта, от формулирования целей задачи и инженерии требований до программирования компонент кода или документации по их использованию.

Многолетняя мировая практика по созданию программных систем позволила накопить практически необозримое количество готовых ресурсов разработки программных систем, использование которых может существенно повлиять на показатели их качества и снижение трудоемкости их создания. Но для этого разработчик должен иметь доступ к информации об имеющихся ресурсах, а также возможность их сопоставления, поэтому одной из центральных проблем разработки программ на сегодня можно считать проблему спецификации ГОР, понятной для их потенциальных пользователей и достаточной для доступа к ним. Проблему понимания можно решить, если упомянутые спецификации выполнить в терминах соответствующей проблемной области (ПрО), а модели спецификации соответствуют типам ГОР и контексту их применения. Тогда все процессы ЖЦ разработки программных систем могут быть организованы под управлением знаний о соответствующей целям разработки ПрО и наличии ГОР, помогающих данному процессу для данной ПрО. При этом модель ЖЦ можно рассматривать как последовательность принятия решений относительно целесообразности использования имеющихся и доступных ГОР или альтернативной реализации разработчиком функций, соответствующих недоступным ГОР. Для реализации такой модели ЖЦ целесообразно использовать онтологическое представление знаний о ПрО и готовых решениях различного типа, накопленных в ее рамках.

1. ИСПОЛЬЗОВАНИЕ ЗНАНИЙ О ПрО КАК ОПРЕДЕЛЯЮЩИЙ ФАКТОР УСПЕХА В РАЗРАБОТКЕ ПРИКЛАДНЫХ ПРОГРАММНЫХ СИСТЕМ

В сообществе разработчиков программных систем фокус профессиональных усилий постепенно смещается от проблем кодирования к вопросам представления знаний о ПрО, которые должны обслуживать программные системы. Раз-

витие инструментальных средств поддержки инфраструктуры программной инженерии и повышение интеллектуальных возможностей универсальных языков программирования, в частности проблемно-ориентированных, обнаружили «слабое звено» в ЖЦ разработки систем — проблему достижения согласованности между заказчиками и будущими пользователями, обеспечения соответствия разработки не только требованиям заказчиков, но и действительным потребностям потенциальных пользователей, которые не всегда совпадают, иначе говоря, нахождения общего языка для разработчиков, заказчиков и пользователей. Усилия по преодолению этой проблемы породили относительно новое направление развития программной инженерии, центральная идея которого — использование модели ПрО на всех этапах ЖЦ как базы для выработки и представления соответствующих решений. Это направление получило название проблемно-ориентированных методов проектирования (Domain-Driven Design — DDD). Под этим термином понимается систематический подход к разработке программных систем для сложных ПрО, который вобрал в себе широкий набор лучших наработок, методов и приемов, полученных как из теоретических исследований, так и из практики [1]. Их ядро позволяет создать базирующийся на модели ПрО общий язык (ubiquitous language) для команды, участвующей в разработке, включая пользователей и заказчиков, как важные факторы успеха разработки. Принципиальным отличием DDD является роль упомянутого выше общего языка как средства общения разных категорий разработчиков, так и средства документирования продуктов высших этапов ЖЦ, цель которого — проведение рефакторинга (улучшения) системы не столько на уровне кода, сколько на уровне базирующихся на ПрО моделей требований, проектирования, архитектуры. При этом модельным классам присваиваются названия и полномочие реальных классов понятий реального мира, которые должны быть достаточными для организации их взаимодействия ради решения возложенных на них задач. В [1] названы ключевые моменты создания общего языка:

- определение ядра сложной ПрО и подчеркивание ее ключевых отличий;
- определение неявных понятий ПрО и преобразование их в явные;
- определение модельных объектов вдоль ЖЦ разработки как базы трассирования требований на объекты кодирования;
- применение паттернов анализа ПрО, в частности предложенных в [2];
- соотнесение паттернов анализа с паттернами проектирования.

Решение приведенных выше задач позволит создать единый язык для диаграмм, документирования и устного обмена мнениями членов команды на протяжении ЖЦ разработки.

Следующим шагом DDD можно считать архитектуру, ориентированную на модель ПрО в виде послойной структуры, где пласт более высокого уровня использует службы, предоставляемые непосредственно низшим пластом, последний не обязан знать о высших пластах, которые пользуются его услугами, а каждый промежуточный пласт «изолирует» низший пласт от высших.

Развивая приведенные выше идеи, в [3] исследуется последовательность процессов разработки, любой из которых подчинен моделям ПрО, и предложена схема расслоения архитектуры: интерфейс пользователя; прикладное применение; проблемная область; инфраструктура разработки.

Коренные проблемы разработки, по мнению автора [3], лежат в инженерии требований. Именно они являются главным источником знаний относительно модели ПрО, путем анализа которых создается на первом шаге эскиз модели ПрО. Рассматривая поочередно выявленные требования, предлагается выработать упомянутый выше общий язык как отображение особенностей ПрО. Модель ПрО подобна диаграммам широко используемого языка моделирования артефактов программной инженерии UML (Unified Modelling Language), но в терминах понятий, присущих лексике пользователя.

Как утверждают многие авторы, использование идеологии DDD существенно повышает эффективность разработок.

2. ОСОБЕННОСТИ ЖЦ РАЗРАБОТКИ ПОД УПРАВЛЕНИЕМ ЗНАНИЙ О ПрО

Управляемая знаниями разработка требует специальной среды, обладающей средствами накопления спектра моделей ПрО и ассоциируемых с ними коллекций ГОР, полезных на определенных этапах ЖЦ разработки. Будем различать три категории ПрО.

1. Прикладные, отражающие производственные интересы специалистов выделенного профиля в рамках определенного человеческого сообщества, например, определенные роли в бизнесе (бухгалтер, менеджер по кадрам и т.п.), определенная область науки, определенное региональное сообщество, определенная фирма.

2. Общесистемные, относящиеся к отдельным процессам программной инженерии на отдельных этапах ЖЦ (определение требований, функциональное и архитектурное проектирование, кодирование, тестирование, документирование, обучение пользователей и т.п.), и/или организационные процессы (в частности, касающиеся разработки и использования программных систем, например лицензионные проблемы, антивирусная защита и авторизация доступа, взаимные трансформации форм представления объектов и т.п.). Будем считать, что такие ПрО относятся к домену программной инженерии. В дальнейшем будем ссылаться на них как SEN (Software Engineering) в отличие от других прикладных областей, за которыми сохранено обозначение ПрО. Система знаний относительно SEN отвечает [4] и играет двойную роль: во-первых, отражает независимые от прикладных применений знания относительно процессов ЖЦ разработки и служит для классификации универсальных методов, инструментов и ГОР, доступных системе на конкретных этапах ЖЦ. Во-вторых, используется для аннотирования ГОР конкретного приложения на конкретном этапе ЖЦ.

3. Область знаний, относящаяся к средствам спецификации различных типов ГОР, известных в определенных кругах профессиональных резработчиков, в том числе допустимые варианты (формы вариантности), которые предоставляет данный тип ГОР [5].

Таким образом, ГОР в системе аттестуются по трем главным измерениям: определенному приложению; определенному этапу ЖЦ разработки; определенному типу ГОР. Высшим уровнем классификации будем считать область приложения. Предложенная категоризация знаний позволяет разработчику получать следующие информационные услуги: 1) поиск знаний, соответствующих ПрО, в сфере которой он ведет разработку; 2) поиск ГОР, которые известны в данной ПрО; 3) поиск ГОР, которые могут использоваться на определенном этапе ЖЦ; 4) поиск известных ГОР заданного типа.

При наличии управления знаниями перечисленных выше трех категорий традиционный ЖЦ разработки можно модифицировать следующим образом:

Этап 1. Инженерия под управлением знаний о ПрО требований на разработку.

Этап 2. Поиск ГОР, которые отвечают минимальному когнитивному расстоянию от текущей стадии разработки, начиная от требований, до стадии, максимально приближенной к завершающей стадии исполнения (напомним, когнитивным расстоянием в программной инженерии называют количество интеллектуальных усилий, необходимых разработчику для преобразования создаваемого продукта при переходе от одной стадии ЖЦ к другой [6]). Далее приведен краткий обзор известных современных типов ГОР, обладающих сравнительно небольшим когнитивным расстоянием.

Этап 3. Принятие решения относительно определения достаточности найденных ГОР или генерация требований на дополнительные ресурсы; при достаточности имеющихся ресурсов — их интеграция, а при недостаточности — предложение пользователю имеющихся инструментов поддержки разработки на текущей стадии ЖЦ.

Наполнение среды разработки знаниями может быть управляемым процессом, который определяется целевой аудиторией (в соответствии с терминологией, предложенной в [1], сообществом ролей пользователей), нужными им ГОР, которые среда может им предоставить, и избранной стратегией их обслуживания. Предполагается, что наполнение среды совокупностью знаний относительно SEN и полезными ПрО должно предшествовать разработке по предложенной модификации ЖЦ.

Из сказанного выше следует, что среда разработки должна содержать методическую, инструментальную, справочную и учебную поддержку информационных потребностей разработчика. Иначе говоря, иметь в своем составе управляемую знаниями систему консультирования на всех этапах ЖЦ относительно доступных ему на текущей стадии его работы ГОР, в том числе методическим правилам и стандартам, придерживаться которых он обязан или заинтересован. Успех такого консультирования базируется на удачных моделях упомянутых выше категорий знаний.

3. ОНТОЛОГИЧЕСКИЕ МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ И ИХ ИСПОЛЬЗОВАНИЕ В РАЗРАБОТКАХ ПРОГРАММ

Большинство современных методов анализа и моделирования ПрО укладываются в известную модель Бангла [7], которая заключается в следующем.

Мир состоит из материальных вещей. Вещи имеют свойства. Свойства делятся на внутренние (для вещей) и взаимные (отношение).

Вещи можно разложить на частицы, которые имеют расширенные свойства по отношению к целому.

Определенное ограничение свойства одной вещи есть закон. Общая форма закона — отнесение свойства одной вещи к свойству другой.

Всякая вещь характеризуется множеством функций состояний, которые отвечают свойствам вещи. Обычно они являются функциями времени, которые определяют индивидуальное свойство вещи в определенный момент времени. Множество функций состояний называется функциональной схемой, или моделью. Одна вещь может описываться несколькими моделями. Состояние вещи определяется как множество значений всех функций состояний отдельной модели.

Состояние, которое может меняться только в результате внешнего взаимодействия, называется стабильным.

Изменение может быть количественным, если меняется значение одного или нескольких свойств, или качественным, если свойства прибавляются или изымаются. Изменение всегда включает изменение состояния определенной вещи. Событием называется каждая пара состояний, входящих в пространство состояний определенной вещи. Событие определяется для пространства состояний одной вещи. Если меняется взаимное свойство двух вещей, существует два события, по одному для каждой вещи. Это есть взаимодействие.

Взаимодействие объектов ПрО в распределенных системах характеризуется известным паттерном взаимодействия GoF [8] и состоит в следующем.

Объект, который порождает событие (так называемый издатель события), публикует сообщение о событии, не зная потенциальных получателей сообщений. Получают сообщение так называемые подписчики событий — объекты, интересующиеся определенными событиями, для которых они заказали (путем контракта с издателем) доставку. Такая парадигма разрешает взаимодействие участников на абстрактном уровне, а именно:

- участники разъединены в пространстве, они не требуют размещения и прямых обращений друг к другу;
- участники разъединены во времени, они не должны быть активными в одно и то же время;
- участники разъединены в потоке передачи данных, получение и отправка данных для одного из них не блокирует других участников.

Контракт между издателем и подписчиком определяет только одно — типы событий, которые вырабатывает издатель.

Описанную выше модель Бангла удобно представить посредством так называемых онтологий [7].

Онтологией принято называть совокупность терминологии, понятий, характерных для них отношений и парадигмы их интерпретации в границах ПрО. Онтология разрешает удерживать пользователя в максимально возможном пространстве предопределенных возможностей, смысл которых зафиксирован и понятен как разработчику, так и заказчику. Очевидно, что пребывание в таком пространстве защищает обе стороны договора (который материализован в виде требований) от взаимных недоразумений.

В соответствии с триадой категорий знаний, предложенной для управления ЖЦ, возникает потребность в отображающих их онтологиях. Заметим, что создание онтологий — это работа, требующая длительного времени и значительных затрат. Необходим также консенсус в обществе: чтобы онтология «работала», нужно чтобы ее признал определенный круг пользователей. Проанализируем предпосылки для создания упомянутых онтологий.

3.1. Онтологии приложений как ГОР. Лучше всего дела обстоят с онтологиями приложений. Следует отметить, что в связи с информатизацией жизни общества растет тенденция к созданию и стандартизации онтологий для значимых ПрО. Широкий набор доступных онтологий упомянут в [8]. Модель ПрО может рассматриваться как один из важнейших готовых ресурсов разработки. Она может отобразить общие и вариантные аспекты и правила варьирования проблем, которые возникают и решаются в границах ПрО. Ее ядро позволяет создать для команды раз-

ных категорий разработчиков, включая пользователей и заказчиков, базирующийся на ее модели общий язык общения. При этом модельным классам присваиваются названия и полномочия реальных классов понятий реального мира, которые должны быть достаточными для организации их взаимодействия ради решения возложенных на них задач.

Структура спецификации ПрО предлагается как онтология, в состав которой входят такие аспекты: объекты ПрО и их ассоциации; состояния; бизнес-функции или варианты использования; имеющиеся ГОР (по типам и по процессам ЖЦ).

Модели ПрО активно используются для многих целей: для коммуникации (между людьми и/или компьютерными системами); для поддержки сохранности информации в компьютерной сфере (внутренние структуры, входы-выходы алгоритмов, поддержка баз терминов); для организации знаний и их аннотирования в целях повторного использования (библиотеки и репозитории); для формулирования требований на разработку или модификацию программного продукта, ориентированного на взаимопонимание между заказчиком и исполнителем заказа; для спецификации аспекта функционального назначения ГОР всех типов.

3.2. Источники для построения онтологий SEN. Построение любой онтологии предполагает наличие устоявшейся системы понятий в рассматриваемой ПрО. Для программной инженерии такая система понятий представлена в [4] фактическим реестром суммы знаний, необходимых соответствующим профессионалам и признаным ими (что очень существенно) как фактический стандарт. Построение соответствующей [4] онтологии требует только технических усилий.

3.3. Источники для построения онтологий ГОР. Средства спецификация готовых ресурсов разработки программных систем для разных типов ГОР существенно отличаются. Современные технологии разработки позволяют использовать широкий спектр типов ГОР для различных контекстов применения и требующих различной квалификации их возможных пользователей. Действительно, ГОР как фрагмент требований на разработку может быть понятен заказчику, тогда как ГОР как фрейм настраиваемого приложения или компонента кода в языке JAVA требует специальной программистской подготовки. Проблема понимания спецификаций ГОР, как всякая проблема моделирования ментальной деятельности, сложна и мало разработана, степень ее формализации должна быть достаточной, чтобы обеспечить точность и однозначность трактования разными носителями интересов в разработке, и в то же время понятной не только математикам, но и способной отобразить детали профессиональных проблем разных слоев специалистов. Однако огромные резервы повторного использования для повышения эффективности разработок стимулировали сосредоточение усилий ведущих разработчиков мира для выработки структуры и правил спецификации наиболее распространенных типов ГОР. В результате появилось два проекта стандартов двух известных компьюрерных концернов: OMG (Object Management Group), предложившего стандарт спецификации широкого спектра ГОР [9], и W3C, предложившего серию стандартов для спецификации веб-сервисов как ГОР массового использования [10–13]. В каждом из названных проектов предлагаются основные аспекты спецификации и иерархическая структура характеризующих их атрибутов.

Общим для проектов является онтологический подход к спецификации аспекта функционального назначения ГОР как перечня дескрипторов с указанием классифицирующих их онтологий, в том числе стандартизованных, таких как онтологии продуктов и услуг, географических названий, математических, химических, медицинских терминов и др. Сходными являются и многие атрибуты, как обобщенные, например, имя, ID (однозначный идентификатор ГОР, поддерживаемый определенным инструментом, например, UUID — Universally Unique Identifier — стандарт Open Software Foundation Group), тип профиля. Могут определяться необязательные атрибуты: дата, состояние, версия, права доступа, неформальное описание, контекст, связи с другими артефактами, правила связывания, вариабельность. Спецификация интерфейсов для веб-сервисов в обоих проектах регулируется общим языком Web Services Definition Language (WSDL) [11], позволяющим определить сетевые адреса связывания и допустимые для них операции ввода-вывода, хотя организационно эта информация регламентируется профилем для веб-сервисов в [9] и отдельным документом в [10]. Сходны атрибуты собственно веб-сервисов (напомним, что проект W3C относится только к веб-сервисам): пространство имен (концепция HTML и XML), услуги, которые предоставляются компонентом или сервисом; услуги, к которым обращается компонент или сервис.

Однако структурно проекты отличаются. Остановимся кратко на особенностях каждого проекта.

В [9] понятие ГОР соответствует термину reusing asset (капитал, имущество, вклад и т.п.). Мы будем употреблять термин ГОР как эквивалент англ. reusing asset. Принципиальным является требование сопровождать собственно артефакт как тело ГОР обязательной спецификацией, названной манифестом ГОР. В нем определяются назначение ГОР и правила его использования, служащие для поиска нужных ГОР и принятия решений об их включении в новые проекты. Иначе говоря, манифест есть аналог поискового образа объекта хранения в библиотеке, а последний может быть, например, элементом требований к разработке или элементом инструкции пользователя системы, элементом архитектуры проекта или его тестом, набором практик (приемов) для улучшения рабочих характеристик продукта или программным кодом реализации его компоненты. Желание охватить все разновидности ГОР, которые могут возникать на всех стадиях разработки, привело к размежеванию базового ядра спецификации, общего для всех типов ГОР, от специфических особенностей ГОР конкретного типа (как продуктов конкретной деятельности на конкретном этапе разработки). Последние составляют так называемые профили спецификации. Профиль есть расширение базового ядра спецификаций (базового профиля) для конкретного типа ГОР. В [9] определены профили для программных компонент и веб-сервисов. Предполагается построить профили для известных типов ГОР (например, компонентов CORBA, Java Beans, компонентов требований, паттернов, фреймов, модулей тестовой инфраструктуры и др.).

Каждый профиль определяет базовую иерархию и сущность характеристик, которыми описываются потребительские свойства ГОР соответствующего типа, следовательно, является источником онтологии атрибутов этого типа.

Базовый профиль имеет обязательные секции спецификации:

- *asset* идентифицирует и классифицирует ГОР;
- *контекст* характеризует принадлежность ГОР к определенной стадии моделирования (бизнес-модель, модель требований или проектирования, компонент кода, теста, документирования;
- *использование* характеризует виды работ (пользователя или инструментального средства), необходимые для использования ГОР, в том числе для настраивания вариантных свойств ГОР;
- *решение* определяет артефакты, соответствующие ГОР, имеет атрибуты: имя, тип (главный тип, например компонент кода, и несколько вторичных типов, которые его объясняют, например соответствующие ему требования, отдельные диаграммы UML, разделы инструкций для пользователя и т.п.), так называемые точки вариантности (концептуальные позиции артефакта, который пользователь имеет возможность изменять после передачи ему артефакта для использования);
- *зависимости* определяет ссылки на артефакты, связанные с данным определенными отношениями (ассоциация, агрегация, наследование, зависимость по изменениям и т.п.).

Заметим, что упомянутые предложения определяют лишь стандартный каркас спецификации, свобода детализации которого остается за разработчиком нужного профиля. В [5] представлен профиль спецификации адаптивных свойств программных продуктов.

Проект W3C выполнен как согласованный набор трех отдельных компонентов спецификации ГОР типа веб-сервисов: [11] регламентирует описание интерфейсов, [12] — стандарты для обмена сообщениями и вызова удаленных процедур, [13] — содержательное описание, ориентированное на понимание функций веб-сервиса. Именно последний документ наиболее близок к [9] по системе используемых базовых понятий, хотя предмет его рассмотрения ограничен ГОР одного типа. Указанное обстоятельство позволяет сделать вывод об устойчивости упомянутых понятий в среде специплистов по программной инженерии и их пригодности как источника онтологии атрибутов ГОР.

3.4. Интеграция онтологий аспектов знаний. Как отмечалось выше, каждому аспекту знаний о ГОР соответствует отдельная онтология, которая может рассматриваться как база знаний ассоциированной с ней фасеты поискового ресурса, поэтому предлагается интеграцию онтологий аспектов проводить не на стадии их представления, а на стадии конструирования под их управлением поисковых образов и поисковых запросов. В частности, при интеграции онтологий ПрО и SEN онтология SEN есть источник так называемых тегов (имен атрибутов ГОР), а онтология ПрО — источник значений таких атрибутов для конкретного ГОР.

3.5. Перечень распространенных типов ГОР с относинельно небольшим когнитивным расстоянием. Модель ПрО как ГОР. Как отмечалось выше, модель ПрО отражает наивысший уровень классификации знаний в управлении раз-

работкой на всех этапах ЖЦ. К характерным для нее ГОР можно отнести следующие: типовые действующие лица-в-роли (актеры) ПрО; типовые задачи ПрО; типовые классы объектов ПрО, их атрибуты и методы; типовые состояния и действия ПрО; элементы типовых требований к разработке; перед-, пост-условия и инварианты; специфические для ПрО методики тестирования; типизированные показатели качества, метрики и требования по качеству; методики достижения качества конечного продукта.

Трансформации моделей спецификации как ГОР. Трансформацией моделей спецификации принято называть автоматическое преобразование входной модели в выходную (целевую) в соответствии с определенными правилами преобразования как для моделей вцелом, так и для одной или совокупности конструкций моделей [14]. Из известных трансформаций с приемлемым когнитивным расстоянием назовем трансляцию с проблемно-ориентированных языков высокого уровня, трансформации UML-диаграмм в коды в системе Rational и подобных ей.

Линейки программных продуктов (Product Lines) как ГОР. Семья продуктов, или линейка продуктов — это множество программных систем, которые разделяют общий управляемый набор качеств, удовлетворяют специфические потребности определенного сегмента рынка, разработаны с использованием общего набора ГОР [15]. Для использования линейки требуется сначала изучить общие свойства создаваемых ею продуктов, а затем доопределить индивидуальные свойства требующегося конкретного продукта линейки.

Языки, специфические для проблемных областей (Domain Specific Languages — DSL) как ГОР. DSL — это своеобразная форма представления знаний относительно модели ПрО, в частности повторно-используемых знаний для решения прикладных задач в границах конкретной ПрО [16]. К повторно-используемым знаниям, встроенным в DSL, в частности, принадлежат выразительные средства языка пользователя — грамматика, форматы выражений, интерфейсы; абстракции модели ПрО; проектные решения реализации модели, а именно архитектура, генераторы кода. Получение вышеупомянутых знаний проводится в тесном взаимодействии с будущими пользователями — профессионалами ПрО. Словарь понятий и онтология определяют, в какой терминологии строить в DSL конструкты вариантности. Общие свойства определяют модели вычислений и примитивы DSL. Альтернативные или необязательные характеристики являются вариабельними и задаются как параметры.

Бизнес-процессы как ГОР. Процессом в бизнесе считается функция получения дополнительной ценности. Тенденция глобализации мировой экономики привела к распространению процессного подхода к управлению бизнесом и соответственно к созданию типичных моделей бизнес-процессов, пригодных для широкого круга бизнес-планов [17]. Такие модели могут трактоваться как готовые ресурсы инженерии требований на разработку программных систем, которые автоматизируют упомянутые процессы.

Артефакты инфраструктуры разработки. К ним относятся специализированные компоненты повторного использования как элементы конфигурации создаваемых систем, а именно: компоненты архитектуры, фреймы, паттерны, диаграммы UML, тесты, тестовые данные, компоненты кода, фрагменты документации и учебных курсов. Другими словами, можно считать, что в качестве ГОР могут выступать как входные данные, так и продукты всех стадий ЖЦ разработки программ (см. табл. 1).

Таблица 1

Аспект разработки	Категории артефактов продуктов
Инженерия требований	Участники формулирования требований Элементы требований Варианты использования Актеры системы Интерфейсы системы
Проектирование	Архитектурные составляющие: спецификации пакетов, компонентов, объектов, их интерфейсы
Кодирование	Спецификация кодов пакетов, компонентов, объектов
Тестирование	Элементы тестов и их данных, варианты интегрированных тестов
Сопровождение (управление изменениями)	Спецификация прогнозируемой вариантности требований для составляющих проекта компонентов кода тестов
Управление конфигурацией	Спецификации выпусков проекта, версий, пакетов, компонентов, объектов, элементов, документации пользователя

Предложенный механизм совместного испольования онтологий ПрО и SEN позволяет создать для всех ролей участников разработки программ совместную понятийную базу знаний. Упомянутая база благодаря своей нормализации посредством онтологий нивелирует последствия недопонимания и неоднозначности, часто привносимые человеческим фактором.

СПИСОК ЛИТЕРАТУРЫ

1. Evans E. Domain-driven design: tackling complexity in the heart of software. — Boston: Addison-Wesley, 2004. — 523 p.
2. Нильсон Дж. Применение DDD и шаблонов проектирования. — М.: ООО «И.Д. Вильямс», 2008. — 560 с.
3. Фаулер М. Архитектура корпоративных программных приложений. — М.: ООО «И.Д. Вильямс», 2006. — 544 с.
4. http:/www.swebok.org.html.
5. Бабенко Л.П. Спецификация прогнозируемой вариантности как инструмент управления изменениями программных продуктов UML // Кибернетика и системный анализ. — 2007. — № 3. — С. 156–163.
6. Бабенко Л.П., Лаврiщева К.М. Основи програмної инженерії. — Київ: Знання, 2001. — 269 с.
7. Gomes-Perez A., Fernandez-Lopez M., Corcho O. Ontological Engineering. — London: Springer-Verlag, 2004. — 403 с.
8. Бабенко Л.П., Поляничко С.Л. Методи нормалізації знань в інфраструктурі розробки програм // Кибернетика и системный анализ. — 2006. — № 1. — С. 167–173.
9. Reusable Asset Specifications (RAS) OMG Avaible Specifications Version 2.2., Date: November 2005. — http://www.omg.org.
10. http.www.w3.org.
11. http://www.w3.org/TR/2004/WD-wsdl20-.
12. http://www.w3.org/TR/2Q03/REC-soapl2-partl-20030624/.
13. http://uddi.org.
14. Mens T., Van Gorp P., Czarnecki K.A. Taxonomy of model transformation. — http://drops.dagstuhl.de/2-5/11.
15. Northrop L.M. Sei's Software Product Line Tenets // IEEE Software. — 2002. — **19,** N 4. — P. 32–39.
16. Mernik M., Heering J., Sloane A.M. When and how to develop domain-specific languages // ACM Comput. Surveys. — 2006. — **37,** N 4. — P. 316–344.
17. Репин В.И., Елиферов В.Г. Процессный подход к управлению. Моделирование бизнес-процессов. — М.: ФИА «Стандарты и качество», 2007. — 408 с.

Поступила 26.06.2008