Разработка и сопровождение DSM-решений на основе MSF*

Д. В. Кознов
dim@DK12687.spb.edu

В связи с проблемами практического использования UML сейчас развивается альтернативный подход — предметно-ориентированное визуальное моделирование (Domain-Specific Modeling, DSM-подход). Активно разрабатываются технологии типа Microsoft DSL Tools, Eclipse/GMF, MetaEdit+ и др., предназначенные для облегчения создания DSM-решений — языков, методов и программных средств, разрешающих проблемы конкретной предметной области, конкретных разработчиков. В данной статье предлагается модель процесса эволюции DSM-решений, создаваемых внутри одной компании — для product lines, проектов по разработке многоверсионных продуктов и проч. Модель ориентирована на малые и средние компании, способные выделить относительно небольшой бюджет на подобные цели, предполагает «целиковый» подход (то есть решение создается в рамках одного проекта, а не через последовательность проектов) и строится на основе модели процессов MSF 3.1, конфигурируя последнюю под особенности данной задачи.

Введение

Стандартные средства визуального моделирования (язык UML, методы типа RUP, CASE-пакеты типа Rational Rose и т. д.) часто требуют значительной адаптации и «подгонки» при практическом

* Исследование выполнено при частичной финансовой поддержке РФФИ (грант 05-0951/07).

использовании. Контекст использования становится более определенным, общие идеи и подходы получается детализировать, формируя удобный и адекватный инструментарий. Во многих случаях эту задачу удобно решать, используя лишь общие идеи и базовые принципы стандартных средств, не пытаясь их настроить, а создавать свои собственные. В этом и заключается суть DSM-подхода. Активно развиваются сейчас такие технологии, как Microsoft DSL Tools [1,2], Eclipse/GMF [3], MetaEdit+ [4] и др., которые позволяют использовать DSM-подход на практике. Можно с уверенностью сказать, что DSM-подход совместно, например, с product line парадигмой разработки ПО [5] способен дать визуальному моделированию «второе дыхание», разрешив многие трудности исполльзования UML, которые обсуждаются, например, в [6]. Создавать собственные языки и программные инструменты их поддержки становится все легче создавать и поддерживать.

В этой статье, вслед за [2, 7], предметно-ориентированное моделирование рассматривается применительно к контекстам, существующим внутри компаний-разработчиков прикладного ПО. При этом создание DSM-решения, удовлетворяющего потребностям компании и решающего какие-то определенные, специфичные задачи ее процесса разработки, оказывается внутренним проектом в компании и поэтому имеет свои сложности[1]. Это — отсутствие «объективной реальности» в виде внешнего заказчика, трудность удержания четкого фокуса разработки, сложности с организацией сопровождения, трудности с созданием удобного визуального языка.

Мы рассматриваем модель разработки и сопровождения DSM-решения, основные идеи которой были изложены нами в [8]. В данной работе мы встраиваем ее в модель процессов MSF 3.1 [9], конфигурируя последнюю под особенности нашего случая. Методология MSF выбрана как базис в силу того, что она ориентируется на разработку решения, а не просто ПО (то есть предполагает внедрение, организацию сопровождения, обучение и создание необходимой документации). Кроме того, модель процесса MSF представляет единый взгляд на разработку и внедрение, является итеративной и водопадной одновременно, жестко ориентирована на выдачу

[1] В этой ситуации, как много лет назад относительно разработки ПО вообще, встают вопросы наличия адекватного процесса, обеспечения качества, надежности и сопровождаемости результатов разработки.

конечного результата[2]. Наконец, MSF хорошо структурирует опыт автора по разработке, внедрению и сопровождению DSM-решений [11–15].

Необходимо отметить, что в разных компаниях (в том числе и в российских) существует не так уж мало DSM-решений. Принося существенную пользу, эти решения, тем не менее, доставляют много хлопот. Во многих случаях необходимо системное в*и*дение всего жизненного цикла таких решений, рассчитываемое на 10 и более лет. Именно для формирования комплексного, системного в*и*дения в данной области и предназначаются методы, предлагаемые в этой работе.

1. Методология MSF

MSF расшифровывается как Microsoft Solution Framework и обозначает технологию разработки ПО, используемую компанией Microsoft для собственных разработок. Стандарт имеет несколько версий (последняя — версия номер 4), а также несколько вариаций. Вцелом его задача — обобщить обширный опыт компании Microsoft по разработке разнообразного ПО в виде определенных положений и рекоммендаций. В «чистом» виде стандарт не используется даже внутри Microsoft. Такая гибкость, недогматичность даелает MSF привлекательным в качестве исходного материала для разработки более специализированных процессов. Мы будем рассматривать версию 3.1 этой методологии.

В данной работе мы рассмативаемаем вариацию MSF, предназначенную для создания IT-решения — скоординированной поставки набора элементов (таких, как программно-технические средства, документация, обучение и сопровождение), необходимых для удовлетворения некоторой бизнес-потребности конкретного заказчика.

Особое внимание в MSF уделяется внедрению IT-решения в бизнес заказчика, включая его передачу группе сопровождения и поддержки. После такой передачи процесс разработки заканчивается, а сопровождение в MSF не входит.

MSF состоит из следующих дисциплин: модель процессов

[2]Эта ориентация, на наш взгляд, более эффективная, чем, например, в моделях RUP [16] или спиральной модели Боема [17], которые определяют полный жизненный цикл ПО, оканчивающийся лишь смертью последнего. При этом разработка определяется так же, как и сопровождение, хотя часто это совершенно разные процессы.

MSF, модель команды, модель управления проектами, дисциплина управления рисками, управление подготовкой (управление знаниями, необходимыми при создании конкретного IT-решения).

Модель процесса MSF объединяет спиральную и водопадную модели. От спиральной берется частый выпуск версий и итеративное наращивание функциональности, постепенная разработка документов проекта. От водопадной модели берутся фазы и вехи, устанавливающие поступательным движением проекта от начала к концу.

MSF разбивает процесс разработки на следующую последовательность сменяющих друг друга фаз.

- *Выработка концепции.* Здесь происходит создание и сплочение проектной группы на основе выработки единого видения. Результатами фазы являются общее описание и рамки проекта (vision/scope document), документ с оценкой рисков (risk assessment document) и описание структуры проекта (project structure document).
- *Планирование.* На этой фазе происходит составление планов проекта — функциональной спецификации, документов и моделей проектирования, рабочих планов, оценок проектных затрат и сроков разработки различных составляющих проекта. Результатами фазы являются функциональная спецификация, план управления рисками, сводный план и сводный календарный график проекта.
- *Разработка системы.* Результатами этой фазы являются исходный и исполнимый код приложений, скрипты установки и конфигурирования, окончательная функциональная спецификация, материалы поддержки решения, спецификации и сценарии тестов.
- *Стабилизация* — тестирование разработанного решения, при этом внимание фокусируется на его эксплуатации в реалистичной модели производственной среды (то есть выполняется пилотное внедрение). Результатами фазы является окончательный продукт, документация выпуска (release notes), материалы поддержки решения, результаты и инструментарий тестирования (последнее — для команды сопровождения), исходный и исполняемый код приложений, проектная документация.
- *Внедрение* решения в бизнес заказчика, стабилизация внедренного решения, передача работы персоналу поддержки и

сопровождения, получение со стороны заказчика окончательного одобрения. Основными результатами фазы являются информационные системы эксплуатации и поддержки, процедуры и процессы, базы знаний, отчеты, журналы протоколов (logbooks), версии проектных документов, массивы данных (load sets) и программный код, разработанные во время выполнения проекта, а также отчет о завершении проекта (project close-out report) и окончательные версии всех проектных документов.

Возникает закономерный вопрос — можно ли использовать модель процессов MSF отдельно от всего остального (модели команды, дисциплины управления рисками и пр.)? Тут можно сослаться на гибкость и масштабируемость MSF, о чем уже говорилось выше.

2. Дополнительные риски

Проекты по созданию DSM-решений имеют следующие дополнительные риски.

Трудность в содании удобных стредств (usability). Те модели, которые можно будет создавать с помощью DSM-решения, должны быть удобны конкретным людям. Такое удобство является очень субъективным фактором, причем эта субъективность значительно выше, чем удобство произвольных IT-решений. Ведь авторы DSM-решений часто являются специалистами квалификации существенно выше средней, склонными к свободнному оперированию абстрактными моделями. Те инструментальные абстракции, которые они воплощают в графических редакторах, часто оказываются совершенно не удобными для рядовых разработчиков, которые сидят в соседней или даже в этой же комнате.

Кто заказчик решения? Обычно заказчик ПО платит за него и знает, что делать со своей покупкой. В случае разработки заказного IT-решения он также обеспечивает ресурсы и исходную благоприятную атмосферу для полноценного взаимодействия разработчиков и будущих пользователей, позволяя состоятся процессу уточнения требований, передаче решения, обучению правилам его эксплуатации, внедрению и проч. В случае внутренних проектов деньги часто платит сама компания. Нет ответственного лица-заказчика, а есть много причастных к проекту лиц, и у всех них свое мнение о проекте, вокруг него много «политики». В итоге трудно формулировать и удерживать фокус разработки.

Трудности управления требованиями. В начале разработки много энтузиазма, «великих» планов. Часто при этом ослабевает чувство реальности: пользователи хотят несбыточного, разработчики, пользуясь удобным случаем, мечтают реализовать «правильную» систему. По ходу разработки постоянно возникают совершенно новые требования. Ситуация, когда мы делаем систему для себя, часто становится источником неразберихи.

Трудности в управлении ресурсами. Как правило, бюджет подобных внутренних проектов в маленьких и средних IT-компаниях объективно скромен — они не могут себе позволить тратить много на внутренние цели. Не хватает и персонала для участия в таких начинаниях: квалификация работников в таких проектах должна быть очень высокой, а такие работники, как правило, заняты в производственном процессе. Текущие бизнес-приоритеты постоянно выдергивают участников из проекта по разработке DSM-решения, а финансирование часто приостанавливается. Ведь если компания начинает испытывать финансовые затруднения, первое, что «режут», — это бюджет таких проектов. Наконец, очень трудно организовать сопровождение DSM-решений, то есть нужно постоянно поддерживать фоновую деятельность в этом направлении с возможностью усиливать ее временами.

Legacy-тенденция. Созданные и внедренные DSM-решения имеют тенденцию превращаться в legacy. Это происходит из-за отсутствия их надлежащего сопровождения и развития, а также из-за устаревания базовых технологий. Перенос DSM-решений на новые технологии часто оказывается запредельным по затратам процессом, а для их минимального сопровождения требуются люди, которые знакомы и согласны работать со старыми технологиями, уже не использующимися на рынке. Кроме того, возникают трудности в интеграции таких решений с современными продуктами.

3. Терминология

Ниже представлен ряд определений, используемых в статье.

Предметная область (problem domain, domain) — это часть реального мира (люди, знания, бизнес-интересы и проч.), объединенная в одно целое для удовлетворения определенных потребностей рынка [7]. Предметная область может быть как достаточно абстрактной — какое-нибудь открытое сообщество, например сообщество разработчиков Linux, так и очень определенной, с ясно очер-

ченными границами. В данной работе будут рассмотрены предметные области, целиком входящие в какую-либо компанию по разработке программного обеспечения: линейка программных продуктов (product lines), процесс разработки многоверсионного и, соответственно, «долгоживущего» продукта, активы стандартного процесса компании, созданного в рамках внедрения CMM (defined process) и т.д.

Предметно-ориентированный язык (Domain Specific Language, DSL) — это язык, который создается для использования в рамках некоторой предметной области. Мы будем рассматривать только визуальные предметно-ориентированные языки, используя для их обозначения аббревиатуру DSL.

DSM-решение — это конкретный DSL, метод его использования, а также соответствующие средства инструментальной поддержки, внедренные в конкретный процесс разработки ПО. Это определение базируется на определении IT-решения в рамках MSF [9], а также на понятии технологического решения [15].

DSM-пакет — это часть DSM-решения, соответствующего программным средствам [19]. Ниже он будет называться также графическим редактором.

DSM-платформа — это инструментальные средства разработки DSM-пакетов (например, Eclipse/GMF, Microsoft DSL Tools) [18, 19].

Метамодель — это формально описанная структура предметной области, ее основные понятия, атрибуты, связи. Также метамодель оказывается основой нового DSL, определяя его конструкции и их взаимосвязи[3].

Концептуальная метамодель — это формальное описание предметной области, выполненное для разработки будущего DSL и предназначенное для обсуждения с экспертами, будущими пользователями DSL, разработчиками, менеджерами и т. д. Здесь удобно использовать «говорящие» (например, русские) идентификаторы, метамодель должна быть компактна и выразительна. Как правило, DSM-платформы не поддерживают концептуальные модели, а предлагают лишь реализационные. Однако метамодель в терминах Microsoft DSL Tools содержит много реализационных деталей, имеет фиксированную структуру и даже для небольших примеров до-

[3]Определенный здесь термин «метамодель» является синонимом термина «абстрактный синтаксис» из [2, 18].

статочно громоздка. Ее невозможно обсуждать с пользователями DSM-решения. Создавая решения на основе Microsoft DSL Tools, мы использовали Microsoft Visio для создания концептуальной модели. Технология Eclipse/GMF также лишена средств для задания концептуальной модели.

Реализационная метамодель — это метамодель DSL, созданная средствами DSM-платформы и предназначенная для автоматической генерации графического редактора. Она оказывается «хребтом» DSM-пакета в различных DSM-платформах, в ней, по сравнению с концептуальной метамоделью, появляется большое количество реализационных деталей. В Microsoft DSL Tools реализационная метамодель монолитна, включает в себя средство задания абстрактного синтаксиса DSL, его нотации и связей между ними. В GMF нужно строить несколько взаимосвязанных моделей разных видов, и вместе они образуют метамодель DSL, по которой генерируется код редактора. Дальнейшие подробности можно найти в [18, 19].

4. Модель процесса

4.1. Как мы изменяем модель процессов MSF

Облегченное начальное планирование. Требования к DSM-решению редко удается определить сразу. Степень неопределенности здесь выше, чем при разработке обычных IT-решений в силу новизны задачи, обилия заинтересованных сторон и проч. дополнительных рисков, о которых речь шла выше. Поэтому управление требованиями и планирование «размазаны» по всему процессу.

Кроме того, видоизменяется проектирование, входящее в фазу планирования MSF. Использование таких технологий, как Eclipse GMF и Microsoft DSL Tools, сильно упрощает разработку архитектуры решения. С другой стороны, появляется важный артефакт проектирования — метамодель DSL.

Большая итеративность разработки. После каждой итерации происходит оценка полученных результатов и определяются требования к следующей итерации. Более того, итерации позволяют сделать проект более устойчивым к сбоям финансирования. Все это близко к спиральной модели Боема [17].

Нет акцента на тестировании и обучении, соответственно

не уделяется время на создание эксплуатационной документации. Это позволяет существенно экономить бюджет проекта. Пользователи вовлекаются в процесс разработки, они же во многом являются и тестеровщиками. Последнее не очень гуманно, но возможно в силу того, что процесс происходит в рамках одной компании.

Включение в модель сопровождения и эволюции — не происходит передачи системы команде заказчика; DSM-решение сопровождается модифицированной командой разработчиков, перешедшей в режим сопровождения — существенно менее затратный для компании. Таким образом, весь процесс делится на *создание* решения (рис. 1), его дальнейшую *эксплуатацию и сопровождение* (рис. 2), а также возможна *модеринизация* — переделка DSM-решения, что происходит не в режиме сопровождения, а путем организации нового проекта.

Однако важно, что благодаря MSF у процесса разработки DSM-решения есть начало (выработка концепции) и конец (стабилизация и внедрение). Иначе подобные проекты имеют тенденцию переходить в бесконечную деятельность с неясными перспективами.

4.2. Создание DSM-решения

На рис. 1 представлена модель создания DSM-решения. Отдельными кубиками внутри фаз MSF обозначены те виды деятельности, которые отличаются от стандартных. Часть стандартных действий MSF в нашей модели вообще отсутствует, о чем было сказано выше. То, что совпадает, мы не рисовали (а совпадает много, поскольку DSM-решение — частный случай IT-решения).

Выработка концепции. Определение рамок проекта имеет первостепенное значение особенно ввиду изменчивости требований к DSM-решению. Необходимо ясно сформулировать ответ на вопрос, что мы создаем? И этот ответ не должен меняться, несмотря на уточнение, добавление и переосмысление требований. Исходя из этого, необходимо определить бюджет всей разработки (с погрешностями, но нужно), а также понять, во что обойдется поддержка уже внедренного решения. Многие проекты по созданию DSM-решений страдают от того, что над такими подсчетами просто никто не задумывается, а потом выясняется, что у компании недостаточно ресурсов. Если по ходу разработки рамки все-таки начинают меняться, то есть проект превращается во что-то иное, то это должно быть зафиксировано, обсуждено со всеми заинтере-

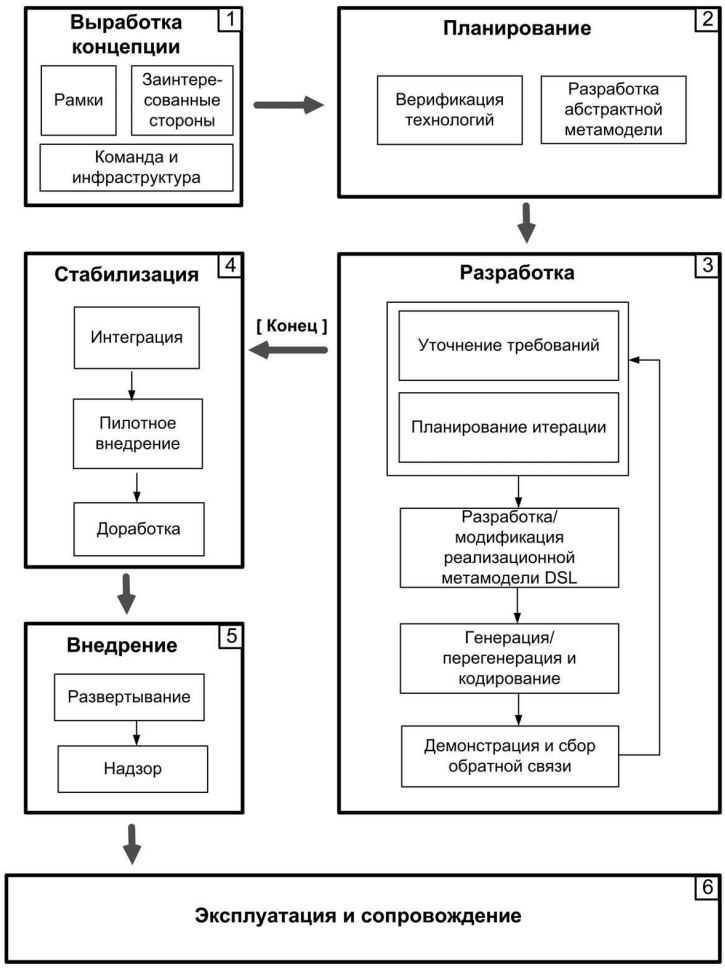

Рис. 1. Модель создания DSM-решения.

сованными лицами и выполнены все нужные действия (вплоть до закрытия проекта). «Ползучесть» рамок приводит к реализации большого количества демо-функциональности, которая не может быть рабочим инструментом.

Важным является определить заказчика DSM-решения. В обыч-

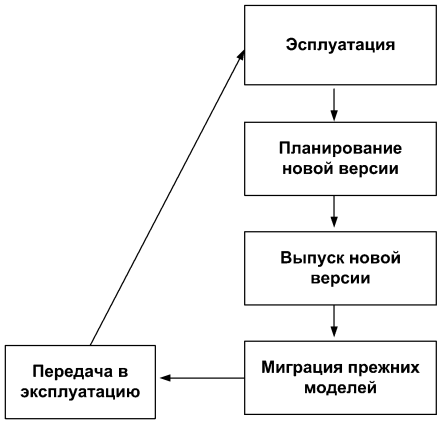

Рис. 2. Модель эксплуатации и сопровождения DSM-решения.

ном IT-проекте заказчик имеет ряд обязательств (подробнее см. [9, 10]). В нашем случае он также отвечает за устойчивость финансирования проекта и за поддержание открытых и доброжелательных отношений между разработчиками и пользователями. Важно, чтобы у него были административные полномочия для того, чтобы влиять и на тех, и на других. Заказчик также имеет решающий голос при создании требований к следующей итерации. Лучше всего, когда заказчик лично заинтересован в проекте, например желает использовать успех проекта для роста по служебной лестнице.

Команда и инфраструктура внутреннего проекта должны быть тщательно определены. Ведь все участники имеют свои рабочие места в разных частях офисов компании и часто — нагрузку, которую с них никто не снимет, несмотря на участие в проекте по разработке DSM-решения. Руководитель проекта должен понять, насколько реально работать с такими людьми, учитывая все особенности их занятости и коммуникаций с ними.

Планирование. Верификация технологий означает выбор подходящей DSM-платформы. Она должна быть перспективной с точки зрения ее развития и сопровождения производителем, чтобы DSM-решение через некоторое время не превратилось в legacy. О разных критериях выбора DSM-платформы см. [18, 19].

Основы метамодели DSL закладываются именно здесь, в начале

проекта. С помощью ее концептуального представления становится возможным обсуждать метамодель с пользователями, возможно, с заказчиком.

Разработка. В рамках этой фазы определяется типовая итерация. Итеративная разработка DSM-решения является основным способом снятия рисков управления требованиями.

Уточнение требований и абстрактной метамодели происходит вместе с пользователями и заказчиком решения на основе демонстрации результатов предыдущей фазы. Одновременно происходит планирование следующей итерации. После этого уточняется абстрактная метамодель, а также меняется реализационная метамодель. Последнее означает как «протаскивание» изменений из концептуальной модели, так и то, что нужно сделать для реализации новой функциональности. В принципе здесь же нужно синхронизировать и концептуальную метамодель с реализационной. Синхронизацию в обе стороны можно осуществить автоматически, как рассказывалось в [8].

Не все особенности DSM-пакета обычно удается задать как свойства реализационной метамодели и сгенерировать автоматически соответствующий программный код редактора. Часть функциональности приходится реализовывать «вручную». Для ее интеграции со сгенерированным кодом в Microsoft DSL Tools используется механизм частичных классов, позволяющий в точно заданных архитектурой DSL Tools местах вставить свой код, который «подвешивается» в реализационную метамодель. В Eclipse/GMF используется другой механизм. Там разработчик помечает особым комментарием методы, которые он поправил руками, и при перегенерации они не затрагиваются. Первый подход имеет минус в том, что разработчики DSL Tools смогли предусмотреть не все потребности в ручном кодировании, но мы пока с таким не столкнулись. Второй подход лишен этого недостатка, но если в реализационной метамодели были изменения, которые должны попасть в исправленные методы, то они туда не попадут.

Демонстрация и сбор обратной связи очень важны для снятия риска разработать пакет, не отвечающий интересам пользователей или просто неудобного. Именно для этого в нашей модели и существуют итерации. В первых версиях результаты только демонстрируются, а пользователи ничего не пробуют сделать самостоятельно: решение еще «сырое», и они получат отрицательное впечатление от его использования. На последних итерациях разработчики реше-

ния могут вместе с пользователями создавать какие-то небольшие примеры из целевой предметной области. При этом важно, чтобы примеры предлагали сами пользователи. Подробно проинструктированные пользователи могут делать что-то самостоятельно. Это также важно, поскольку они могут дать ценную обратную связь. Как правило, к этому времени еще не готов инсталляционный пакет, и, чтобы не затруднять пользователя проблемами развертки, все эксперименты должны происходить на полностью развернутом где-то у пользователей решении.

Стабилизация. Основной акцент здесь делается на следующем.

Часто DSM-решения имеют специфические требования по интеграции и/или по окружению, в котором они должны/могут функционировать. Например, разработанное нами решение на базе Microsoft MS Visio и Visual Studio должно было работать без Studio [13]. Создание и тестирование автономных инсталляционных пакетов часто оказывается непростой задачей.

Деятельность по пилотному внедрению решения присутствует в MSF и выделена в нашей модели особо из-за ее высокой важности в процессе разработки DSM-решения. Пользователи, наконец, пробуют его внедрить в какой-нибудь некритический производственный проект. При этом они, как правило, находят много ошибок, а также могут сформулировать требования к доработке решения. Наличие итераций и вовлеченность пользователей в процесс разработки должны снять риск того, что пользователи при пилотном внедрению скажут, что все совсем плохо.

По результатам пилотного внедрения происходит доработка — исправление найденных ошибок и реализация дополнтельной функциональности.

Внедрение. Основой акцент здесь делается на следующем.

Решение разворачивается на рабочих местах пользователей. Если его системное обслуживание требует дополнительных усилий, то системным администраторам передают нужную информацию. Тут же происходит и обучение, если оно необходимо. Однако мы надеемся на самообучение пользователей, так как они являются программистами и часть из них участвовала в создании решения. Тем не менее ответы на вопросы, презентации или даже лекции могут понадобиться.

Первое время после внедрения необходим надзор за решением: разработчики должны быть готовы к разным неожиданностям

главным образом в виде сложных ошибок, вдруг возникших и парализовавших использование решения. В этом случае разработчики должны оперативно исправить ошибки и выдать пользователям новую версию решения. Оперативность здесь важна, чтобы нанести минимальный ущерб производственному процессу. Время, в течение которого команда разработчиков еще получает зарплату за участие в проекте, определяется совместно руководителем проекта и заказчиком. После того как они решат, что решение стало достаточно стабильным, команда сокращается до минимума, необходимого для сопровождения и поддержки решения.

4.3. Эксплуатация и сопровождение

При эксплуатации и сопровождении DSM-решения ключевым фактором является постоянно действующая команда. Этой командой может быть и один человек — главный автор решения, которому придаются, по необходимости, дополнительные ресурсы. Важно, чтобы DSM-решение постоянно «жило»: исправлялись ошибки, реализовывалась новая функциональность и, что особенно важно, оно должно своевременно переноситься под новые версии DSM-платформы, чтобы не превращаться в legacy. При этом важно уделять должное внимание вопросам миграции прежних моделей пользователей [20]. Сопровождение лучше организовать как последовательность новых версий, не допуская замены отдельных бинарных файлов с высылкой обновлений по электронной почте. С другой стороны, тут важно не переусердствовать в формализме — связь авторов и пользователей должна быть живой, открытой, не обременительной ни для одной из сторон. Процесс эксплуатации и сопровождения схематично изображен на рис. 2.

4.4. Модернизация

Успешные DSM-решения эксплуатируются годами, иногда — десятилетиями. Это значит, что они устаревают. В частности, устаревают генерационные DSM-решения, которые генерируют целевые приложения или их фрагменты по моделям. Ведь платформы целевых систем эволюционизирую. Тут уж без модернизации не обойтись.

Важно отметить, что модернизация является очень трудоемким процессом, и его лучше не проводить в фоновом режиме, а снова

организовывать проект. Этот проект уже не имеет многих из тех рисков, что при исходном создании DSM-решений (например, проблем с usability), ему уже не обязательно быть таким итеративным. Поэтому его можно проводить по обычной схеме разработки ПО. Попытки делать модернизацию в фоновом режиме часто приводят к провалам и укреплению legacy-тенденции.

Заключение

Представленная в работе модель процесса отражает опыт автора и не претендует на всеобщность. В контексте DSM-методологии, предложенной в работе [19], может существовать несколько принципиально разных моделей процесса. В частности, данная модель ориентируется на «скудные» DSM-решения, которые разрабатываются в маленьких и средних компаниях по разработке ПО. В крупных компаниях ресурсов на создание DSM-решения может быть отпущено значительно больше — соответственно, будет эффективна другая модель процесса разработки. Модель также предназначена для «целиковой» разработки DSM-решений, в рамках одного проекта, а в действительности таких проктов может быть несколько.

Дальнейшим шагом по детализации и уточнению предложенной модели могло бы быть создание соответствующего шаблона в Microsoft Team Studio. Это позволит эффективно использовать данную модель при разработке DSM-решений в Microsoft Visual Studio, на основе DSL Tools.

Список литературы

[1] Microsoft DSL Tools, in Visual Studio, 2005, SDK. http://www.mycrosoft.com/downloads.

[2] *Greenfield J., Short K. et al.* Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons. 2004. 666 p. (Русский перевод: *Гринфилд Д. и др.* Фабрики разработки программ (Software Factories): потоковая сборка типовых приложений, моделирование, структуры и инструменты. Изд-во «Вильямс», 2006. 592 с.)

[3] Eclipse/GMF. http://www.eclipse.org/gmf/.

[4] *Kelly S., Lyytinen K., Rossi M.* MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment. Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems En-

gineering, Lecture Notes in Computer Science 1080, Springer-Verlag, 1996. P. 1–21.

[5] *Clements P., Northrop L.* A Framework for Software Product Line Practice. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. 2006. http://www.sei.cmu.edu/plp/framework.html.

[6] *Demeyer S., Ducasse S., Tichelaar S.* Why unified is not universal // UML shortcomings for coping with round-trip engineering. Proceedings of UML'99 (The Second international Conference on The Unified Modeling Language), Kaiserslautern, Germany, October 1999 (Lecture Notes in Computer Science, vol. 1723), Rumpe B (ed.). Berlin: Springer, 1999.

[7] *Czarnecki K., Eisenecker U. W.* Generative programming: Methods, Tools, and Applications. Addison-Wesley, 2000, 832 p.

[8] *Кознов Д. В.* Модель разработки и сопровождения решений в области предметно-ориентированного визуального моделирования // SEC(R) 2007.

[9] Microsoft Solutions Framework, Process Model, Version 3.1. 2002. http://www.microsoft.com/msf (Русский перевод: Microsoft Solutions Framework. Модель процессов, версия 3.1. 41 с.)

[10] Microsoft Solutions Framework, Project Managment. Version 3.1. 2002. http://www.microsoft.com/msf (Русский перевод: Microsoft Solutions Framework. Дисциплина управления проектами, версия 3.1. 23 с.).

[11] *Парфенов В. В., Терехов А. Н.* RTST — технология программирования встроенных систем реального времени // Системная информатика. Вып. 5: Архитектурные, формальные и программные модели. Новосибирск, 1997. С. 228–256.

[12] *Ivanov A., Koznov D.* REAL-IT: Model-Based User-Interface Development Environment // Proceedings of IEEE/NASA ISoLA 2005 Workshop, Maryland, USA, 23–24 September 2005. P. 31–41.

[13] *Кознов Д. В., Перегудов А., Бугайченко Д. и др.* Визуальная среда проектирования систем телевизионного вещания // Системное программирование. Вып. 2 / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2006. С. 148–176, http://www.sysprog.info.

[14] *Романовский К. Ю., Кознов Д. В.* Язык DRL для проектирования и разработки документации семейств программных продуктов // Вестн. С.-Петерб. ун-та. Сер. 10. 2007. № 4.

[15] *Кознов Д. В.* Визуальное моделирование компонентного ПО: Дис. ... канд. физ.-мат. наук. СПбГУ, 2000. 82 с.

[16] *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения / Пер. с англ. СПб.: Питер, 2002. 492 с.

[17] *Boehm B.* A Spiral Model of Software Development and Enhancement // IEEE Computer. Vol. 21. N 5. May 1988. P. 61–72.

[18] *Кознов Д. В.* Визуальное моделирование: теория и практика: Учебное пособие. М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2008. 246 с.

[19] *Павлинов А., Кознов Д., Перегудов А. и др.* О средствах разработки проблемно-ориентированных визуальных языков // Системное программирование. Вып. 2 / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2006. С. 121–147. http://www.sysprog.info.

[20] *Иванов А. Н.* Механизмы поддержки циклической разработки ИС в рамках модельно-ориентированного подхода // Системное программирование / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2004. С. 101–123. http://www.sysprog.info.