Эволюция общих активов в семействе средств реинжиниринга программного обеспечения

Т.Н.Попова
tnp@tepkom.ru

Д.В.Кознов
dim@tepkom.ru

А.Е.Тиунова
aet@tepkom.ru

К.Ю.Романовский
kostet@tepkom.ru

Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

Аннотация

Разработка ПО в виде семейства программных продуктов — это подход, позволяющий увеличить эффективность за счет планируемого повторного использования различных активов разработки. В данной работе рассматривается эволюция переиспользуемых активов семейства продуктов для реинжиниринга ПО — Rescueware WorkBench. Рассматриваются три вида общих активов: процедуры и соглашения конфигурационного контроля, архитектура, программные компоненты. Вводится таксономия вариативности для переиспользуемых компонент.

Введение

Увеличение накала конкурентной борьбы на рынке производства программного обеспечения (ПО), а также дальнейшая экспансия программного обеспечения в различные области человеческой деятельности приводит к необходимости повышать продуктивность компаний, занимающихся созданием ПО. Одним из путей к этому является организация производства ПО в виде *семейства программных продуктов (software product line)* — группы программных систем, обладающих общим набором свойств,

© Т.Н.Попова, Д.В.Кознов, А.Е.Тиунова, К.Ю.Романовский, 2004

удовлетворяющих нуждам определенного сегмента рынка и разрабатываемых на основе общих активов в рамках единого технологического процесса [12].

В настоящее время этот подход интенсивно развивается: два консорциума — SEI[1] в США и ITEA[2] в Европе — аккумулируют, обобщают и распространяют успешный промышленный опыт в этой области. Привлекательность метода разработки ПО на основе организации семейства заключается в объединении бизнес-стратегии компании и процесса разработки ПО: стратегия компании на рынке (создание новых продуктов, сопровождение и развитие уже существующих) ставится в прямую зависимость от возможностей переиспользования затрат. То есть от развития одиночных продуктов компании стремятся перейти к совместной разработке и продвижению нескольких продуктов, повышая экономичность и эффективность разработки и, тем самым, получая возможность занять на рынке более широкую область, быстрее реагировать на новые запросы, повышать качество продуктов.

Одним из ключевых вопросов при организации и поддержке семейства программных продуктов являются *общие активы (core assets)* — то, что переиспользуется в разработке нескольких продуктов [12]. При организации семейства программных продуктов прежде всего необходимо, чтобы удельный вес общих активов был достаточно велик — в противном случае продукты дешевле разрабатывать изолированно друг от друга.[3] В то же время, поскольку отдельные продукты должны удовлетворять специфическим требованиям разных заказчиков, они могут значительно отличаться друг от друга, в рамках семейства должна поддерживаться вариативность.

В данной работе мы рассмотрим семейство средств реинжиниринга ПО RescueWare, развиваемое в течение 5 лет совместно компаниями Relativity Technologies (США) и ЗАО "ЛАНИТ-ТЕРКОМ" (Россия). Общий объем работ, выполненных в рам-

[1] Software Engineering Institute, http://www.sei.cmu.edu/plp

[2] Information Technology for European Advancement, http://www.itea-office.org

[3] Необходимо также, чтобы были организационно-технологические возможности для переиспользования общих активов: соответствующая политика в проекте и в организации, подкрепленная менеджментом и корпоративными стандартами; развитая коммуникационная система (разрабочики должны знать, что в компании уже сделано), доступная документация, средства поиска и стандартные процедуры модификации переиспользуемых программных компонент и т.д.

ках этого проекта, к настоящему времени составил около 350 человеко-лет. Основными продуктами этого семейства являются средства автоматизированного анализа и реинжиниринга приложений, созданных на языках COBOL, PL/1, Adabas/Natural, с возможностью переноса этих приложений на современные платформы — C++, Java, Visual Basic [6]. Основная идея этого семейства — построение многоязыковой среды реинжиниринга ПО как единого транслятора с M входными и N выходными языками на основе унификации внутреннего представления структур данных и алгоритмов[4].

Мы прошли путь от разработки одиночных продуктов и создания отдельных переиспользуемых активов к организации полноценного семейства. Это позволило нам эффективно переиспользовать работу нескольких специализированных и территориально распределенных команд разработчиков, гарантировать выпуск качественной продукции, достичь существенной экономии ресурсов.

В данной работе мы подробно рассмотрим эволюцию общих активов этого семейства — процедур и соглашений конфигурационного управления, архитектуры, программных компонент. Мы остановимся также на проблемах, с которыми мы столкнулись, расскажем о способах их преодоления и о дальнейших планах развития семейства RescueWare.

1 История семейства RescueWare

История семейства средств реинжиниринга RescueWare начинается с 1997 г., когда от известной американской компании SEER Technologies (США), занимавшейся разработкой третьей в мире по объему продаж CASE-системы SEER*HPS (High Productivity System) [14] для создания многоплатформенных корпоративных информационных систем, отделилась компания Relativity Technologies (США). Если в компании SEER Technologies задачи реинжиниринга рассматривались в контексте конвертации ПО с различных платформ в SEER*HPS, то компания Relativity Technologies стала развивать средства анализа и реинжиниринга для приложений платформы IBM mainframe, созданных на

[4]Эта идея известна довольно давно — см., например, [1].

языках COBOL, PL/1, Adabas/Natural и некоторых других, с возможностью трансформации старых приложений на современные платформы, включая решение проблемы компонентизации исходных монолитных приложений [6]. Основная группа разработчиков RescueWare была сосредоточена в ЗАО "ЛАНИТ-ТЕРКОМ" (Россия, Санкт-Петербург), образованном на базе Санкт-Петербургского государственного университета.

Первые варианты RescueWare, еще в составе компании SEER Technologies, были созданы на платформе OS/2 в виде набора одиночных продуктов. Эти системы не получили активного коммерческого распространения. Однако на их примере стало понятно, что для эффективного практического использования система реинжиниринга должна быть богатой интерактивной средой [5], а не одиночным языковым конвертором. С другой стороны, разработчики и менеджеры осознали, что разработка одиночных продуктов является слишком дорогой. Были созданы первые общие активы — единый многооконный пользовательский интерфейс, универсальные алгоритмы потокового анализа, единая генерация кода приложения. Тогда же сформировались основные идеи общей архитектуры таких продуктов. Однако в то время они были реализованы лишь частично из-за относительной простоты этих первых продуктов: это были средства для конвертации приложений на COBOL и CSP[5] в SEER*HPS.

В 1997 г., одновременно с образованием компании Relativity Technologies, было принято решение о переносе RescueWare на платформу Microsoft Windows. Одновременно в качестве главного приоритета была обозначена разработка средств анализа и реинжиниринга приложений на языке COBOL. В это же время сложилась основная концепция реинжиниринга, положенная в дальнейшем в основу всех продуктов данного семейства [6, 4]. Важная особенность новой концепции — предоставление возможностей для участия человека в процессе анализа и преобразования устаревших приложений.

В итоге была создана архитектура типового продукта семейства, которая была реализована для его первого представителя.

[5]CSP (Cross System Product) — это CASE-система производства компании IBM, предоставляющая единую среду разработки информационных систем (от спецификации требований до тестирования конечного кода), интегрирующую различные средства работы с данными и разработки пользовательских интерфейсов, использующиеся на mainframe-платформе и для OS/2 [13].

Этот представитель в течение 2-х лет был единственным в семействе.

В 1999 г. началась разработка двух новых представителей семейства — продуктов для анализа и реинжиниринга приложений с языков PL/1 и Adabas/Natural. Проекты стартовали как "пилотные", но через год ввиду успешности их разработки и открывшихся бизнес-перспектив они получили "законный" статус наряду с COBOL-продуктом. За 3 года своего существования новые представители стали с ним сравнимы по функциональности, хотя ресурсов на их разработку было затрачено значительно меньше. Их разрабатывали команды по 6-10 человек в течение 3-х лет, в то время как реализацией родовой архитектуры и разработкой COBOL-продукта занималась команда в 50 человек в течение 5-ти лет.

В первые 2 года был создан также и самостоятельный продукт, предназначенный для анализа и реинжиниринга приложений SEER*HPS в COBOL, который не вошел в семейство. Впоследствии появились другие подобные продукты, например средство конвертации приложений ADW[6] в COBOL. Причина разработки этих продуктов вне семейства заключалась в том, что они в качестве "входа" имели не тексты на языках программирования, а репозитории CASE-систем, что не укладывалось в архитектуру типового представителя семейства. Тем не менее, указанные продукты разрабатывались в этой же инфраструктуре процесса, используя единые процедуры конфигурационного контроля, обеспечения качества и т.д.

В 2000 г. на основе общих активов был реализован продукт по переводу COBOL в COOL:Gen[7]. Эта работа заняла очень небольшое время, в течение которого было выпущено две версии продукта.

[6]ADW (Application Developing Workbench) — это CASE-система, принадлежащая в настоящее время компании Computer Associates International, предназначавшаяся, как CSP и SEER*HPS, для корпоративной разработки и сопровождения многоплатформенных информационных систем [7].

[7]COOL:Gen — это CASE-система, принадлежащая в данный момент компании Computer Associates International, предназначенная для разработки многоплатформенных распределенных приложений и охватывающая все этапы разработки, от планирования до порождения исходного кода приложений и интеграции [8].

2 Общие активы

В RescueWare реализованы следующие типы общих активов:

1. процедуры и соглашения конфигурационного управления;
2. архитектура;
3. переиспользуемые компоненты.

2.1 Процедуры и соглашения конфигурационного управления

Цели конфигурационного управления при создании ПО, согласно [15], заключаются в обеспечении целостности продуктов проекта на протяжении всего его жизненного цикла. Особенности конфигурационного управления проекта RescueWare подробно изложены в работах [16, 3]. В Приложении приведены наиболее важные[8], с точки зрения поддержки семейства ПО, процедуры и соглашения конфигурационного управления проекта RescueWare.

2.2 Архитектура

Архитектура ПО, согласно [9], — это его структура, т.е. программные компоненты, отношения между ними, а также внешне видимые свойства этих компонент. В рамках семейства программных продуктов обычно выделяют типовую архитектуру продукта (reference architecture) — описание абстрактной архитектуры, которая реализуется в каждом продукте семейства [11]. Существует также архитектура конкретных продуктов. В контексте обсуждения общих активов мы будем рассматривать типовую архитектуру продукта семейства RescueWare.

Стратегия работы с устаревшим приложением на основе продуктов RescueWare: анализ устаревшего приложения (legacy understanding), извлечение бизнес-правил и логической структуры системы (knowledge mining), перенос приложения на новую платформу (legacy transformation) — в качестве основных платформ были выбраны Microsoft Visual C++, Microsoft

[8] В соотвествии с классификацией SEI [15].

Visual Basic, Java [6, 4]. Эта стратегия определила общий функциональный состав продуктов семейства, а также позволила им использовать общий пользовательский интерфейс.

Схема типового продукта семейства: интегрированная оболочка (front end), пакетные компоненты (back end), визуальные инструменты (tools). Эта схема точно соответствовала территориальному разделению команды разработчиков: интегрированная оболочка разрабатывалась в Кэри (Северная Каролина, США), пакетные компоненты в Санкт-Петербурге (Россия), визуальные инструменты — в Новосибирске (Россия). В рамках этой схемы существовали как переиспользуемые, так и уникальные для каждого продукта программные компоненты.

Типовой набор компонент. Сюда кроме стандартных для компиляторов компонент (видонезависимый/видозависимый анализ, синтез промежуточного представления, генерация конечного кода, поддержка периода исполнения) вошли репозиторий, диаграммные визуализаторы для различных аспектов приложений, средства гипернавигации по исходным текстам приложений, средства извлечения бизнес-правил из исходных текстов программ и трансформации их в компоненты на современных платформах, а также общие вспомогательные библиотеки [4, 5].

Механизм взаимодействия интегрированной оболочки и пакетных компонент. Любая пакетная компонента в качестве входной информации получает один или несколько элементов репозитория. Результатом ее работы является также изменение репозитория. Запуск компонент из интегрированной оболочки происходит через унифицированный скриптовый механизм, предоставляющий гибкий способ спецификации параметров запуска компоненты, а также последовательности шагов по обработке команды (соответствующей пункту меню, кнопке на панели управления и т.д.). Результат действия компоненты представляется в виде скрипта, который меняет содержимое репозитория. Такой механизм позволяет свести к минимуму затраты на интеграцию новой компоненты, поскольку для этого не требуется никаких изменений в исходном коде интегрированной оболочки.

Единые внутренние интерфейсы. Кроме интерфейса взаимодействия между интегрированной оболочкой и пакетными компонентами существует много иных стандартных интерфейсов: к репозиторию, к единому языково-независимому внутреннему представлению приложения, интерфейс настройки компоненты гипернавигации по исходным текстам приложения и т.д.

2.3 Программные компоненты

Выделим два типа повторного использования компонент — *непосредственное* ("as is", т.е. переиспользование осуществляется без каких-либо модификаций) и *вариативное* (переиспользуемый актив может быть модифицирован). В связи с разнообразием функциональности продуктов семейства (каждый продукт имеет своих заказчиков, выдвигающих специфические требования) особенно важно наладить планируемое и контролируемое вариативное повторное использование компонент, что позволит избежать раскопирования кода с последующей модификацией.

В рассматриваемом семействе применяются следующие виды вариативного повторного использования:

1. *Настраиваемое повторное использование.* Компонента не меняется, вариативность достигается за счет использования средств настройки — инициализационных файлов, специальных скриптов и т.д.

2. *Многоярусное повторное использование.* Программная компонента разделяется на несколько подкомпонент, часть из которых переиспользуется непосредственно, а остальные создаются для каждого представителя индивидуально.

3. *Повторное использование с насыщением.* Функциональность повторно используемой компоненты дополняется в интересах каждого представителя, при этом становясь все полнее. Таким образом, чем больше продуктов выпущено в рамках семейства, тем более полными будут такие компоненты и тем меньше их нужно будет дополнять в интересах новых продуктов.

4. *Повторное использование с расширением.* Функциональность повторно используемой компоненты дополняется в

интересах каждого продукта, причем практически независимо от предыдущих продуктов. Основное отличие от предыдущего типа — насыщение невозможно.

Далее, с учетом приведенной классификации опишем основные переиспользуемые компоненты семейства RescueWare.

Единая интегрированная оболочка исторически была первым общим активом семейства RescueWare. Она предоставляет пользователю интерфейс работы с продуктами семейства и интегрирует их функциональность. Поскольку требуется создавать различные поставки продуктов семейства (в том числе и различные сочетания нескольких совместно работающих продуктов), то интегрированная оболочка имеет гибкие средства настройки пользовательского интерфейса и подключения соответствующих пакетных компонент и визуальных инструментов. Эта настройка происходит при установке инсталляционного пакета — соответствующие скрипты заполняют структуры данных, определяющие внешний вид интегрированной оболочки и связь элементов интерфейса с нужными пакетными компонентами и визуальными инструментами. Единая интегрированная оболочка — это актив, предназначенный для настраиваемого повторного использования.

Репозиторий (со средствами визуализации). Собранная средствами RescueWare информация об анализируемых приложениях хранится в едином репозитории. Реализация этого репозитория и средства его просмотра переиспользуются всеми продуктами семейства. Модель репозитория допускает расширение для различных продуктов, т.е. репозиторий — это компонента, предназначенная для повторного использования с расширением.

Визуальные инструменты: отчеты по инвентаризации анализируемых приложений, отчеты по оценке трудозатрат на реинжиниринг приложения, средство гипернавигации по исходным текстам приложения [2], web-отчеты результатов анализа приложения и т.д. Эти инструменты переиспользовались всеми продуктами семейства, в редких случаях для продукта создавались специализированные инструменты. Визуальные инстру-

менты предназначены для многоярусного повторного использования: собственно визуальное представление данных используется неизменно, а код подготовки самих данных специфичен для каждого продукта.

Единая генерация приложений на выходных языках на основе единого языково-независимого представления приложений на различных входных языках [6, 4]. Такой подход позволил развивать единую генерацию для разных продуктов семейства. Однако эта часть активов была наиболее бурно развиваема при разработке отдельных продуктов семейства, т.е. единая генерация — это актив с насыщаемым повторным использованием.

Компоненты обработки вспомогательных mainframe-языков: языки описания данных и экранных форм (SQL/DDL для DB2 и Oracle, BMS, AS400, DMS и т.д.), встроенные языки — CICS, SQL/DML. При разработке второго и третьего продуктов (они разрабатывались практически одновременно) из этих компонент вычищались зависимости от первого продукта. В итоге они стали настраиваемыми повторно используемыми компонентами.

Компонента тарификации: осуществляет учет действий (синтаксический анализ файла, извлечение бизнес правил, генерация целевого кода и т.п.), выполняемых пользователем с помощью RescueWare. Чем больше действий выполняет пользователь и чем сложнее эти действия, тем дороже ему будет обходиться эксплуатация RescueWare. Эта компонента является настраиваемой повторно используемой.

Служебные библиотеки: работа со строками, файлами, деревьями и т.д. Служебные библиотеки предназначаются для непосредственного повторного использования.

3 Особенности эволюции общих активов

Процедуры и соглашения конфигурационного управления потребовали лишь небольших доработок при переходе от одиноч-

ного продукта к семейству, поскольку эти активы еще при разработке первого продукта достигли высокого уровня зрелости. Этому способствовал распределенный характер разработки и многоверсионность первого продукта (в отдельные моменты поддерживалось одновременно до 3-х версий).

Специально для семейства была создана единая автоматическая система пакетирования продуктов. Средства сборки и тестирования были лишь переконфигурированы — добавлена возможность сборки новых проектов, созданы новые списки рассылки и т.д.

Архитектура типового представителя семейства в основном была спроектирована перед началом разработки и впоследствии менялась незначительно. Такая стабильность объясняется: а) опытом создания одиночных продуктов аналогичного назначения; б) предварительным созданием прототипа семейства; в) однородностью системной инфраструктуры устаревших приложений — IBM mainframe со стандартным набором встроенных языков.

На начальных этапах разработки второго продукта возникали сложности с переиспользованием компонент, поскольку архитектура типового представителя семейства была реализована для одного продукта, а новые продукты появились только через 2 года. Все это время чистота реализации архитектуры не контролировалась, в результате чего в общих активах накопилось много зависимостей от первого продукта и почти все переиспользуемые компоненты требовали “вычищения” этих зависимостей. Некоторые предварительно спроектированные архитектурные решения были реализованы только при создании второго и третьего представителей семейства, например генерация по единому языково—независимому внутреннему представлению приложений на различных выходных языках. Перевод первого продукта на вновь созданные общие активы был трудоемкой задачей. Однако необходимо отметить, что потребовалась лишь локальная работа по реализации, а не масштабные изменения.

В процессе разработки второго и последующих продуктов семейства RescueWare возникло несколько непредвиденных ситуаций:

1. **Незапланированные накладные расходы на адаптацию общих активов.** При составлении планов менедж-

мент полагал, что общие активы, в частности, переиспользуемые программные компоненты, можно использовать без модификации.

2. **Незапланированные накладные расходы на разработку специфичных активов.** Часть активов, которые считались общими, таковыми не оказались, что привело к серьезным издержкам. В частности, оказалось, что некоторые трансляторные техники, применимые к языку COBOL, неприменимы к PL/1; недостаточно была исследована вариативность требований для второго и третьего представителей — то, что было естественно для работы с COBOL-приложениями, оказалось неприменимым к программам на PL/1, и наоборот, последние потребовали реализации существенной дополнительной функциональности, неучтенной в первоначальных планах.

3. **Конфликты группы сопровождения общих активов с остальными группами.** Каждый продукт имеет собственные приоритеты и сроки выхода. При этом группа поддержки общих активов не в состоянии одновременно удовлетворить потребности всех продуктов, вследствие чего возникают конфликты. На этот факт указывается и в других работах, например в [10].

4. **Необходимость существенно видоизменить процедуру инсталляции продуктов после появления второго продукта.** Из-за необходимости одновременного сосуществования нескольких продуктов семейства на одном рабочем месте потребовалось создать единый инсталляционный пакет для всех продуктов семейства. Это оказалось существенной работой, которая не была учтена при планировании.

5. **Трудности асинхронного выпуска различных продуктов.** Из-за необходимости одновременного сосуществования нескольких продуктов семейства на одном рабочем месте различные продукты должны были работать на одной и той же версии общих активов, в результате чего практически не удалось наладить асинхронный выпуск отдельных продуктов семейства.

Заключение

Семейство развивалось в условиях одновременного освоения рынка и острой конкурентной борьбы, поэтому разработка всех продуктов велась параллельно с поиском заказчиков, уточнением требований к этим продуктам, созданием дополнительной функциональности, которая, по мнению маркетологов и менеджеров по продажам, должна обеспечить победу этим продуктам над конкурентами. Это сильно затрудняло разработку, приводило к созданию некоторого количества "тупиковой" функциональности. С другой стороны, такая стратегия оказалась оправданной и привела к созданию зрелых продуктов именно в силу зрелости и стабильности процесса разработки, что во многом было достигнуто благодаря организации семейства.

В итоге удалось достигнуть высокого уровня переиспользования программного кода. Например, для PL/1-продукта общие активы составляют: пакетные компоненты — 65% (менее 10% — насыщаемые компоненты, остальные — либо настраиваемые, либо переиспользуемые непосредственно, либо многоярусные), интегрированная оболочка — 100%, визуальные инструменты — 90%. Однако необходимо отметить, что для каждого следующего за первым представителя семейства пришлось проделать значительную работу по соответствующей модификации общих активов. При этом везде, где это было возможно, упор делался на обобщение сходной функциональности в рамках существующей архитектуры.

Нам не удалось достигнуть полной стабильности для некоторых переиспользуемых программных компонент. Основными причинами этого был интенсивный поток изменяющихся требований (в силу острой конкурентной борьбы на рынке), недоработанность стратегии превращения демонстрационного прототипа в функционально-полноценный продукт, а также организационные противоречия, вызванные переходом к матричной структуре управления.

За время существования семейства было выпущено следующее количество версий:

1. для COBOL-продукта — 12 за 5 лет (из них 9 за последние 3 года);

2. для PL/1-продукта — 4 за 3 года;
3. для Adabas/Natural — 5 за 3 года;
4. для COOL:Gen-продукта — 2 за 3 года.

Это вряд ли удалось бы сделать без семейства, т.е. при разрозненной разработке продуктов.

В качестве дальнейших направлений развития семейства RescueWare можно выделить организацию повторного использования пользовательской документации и дальнейшее развитие системы пакетирования для возможности поставки многочисленных и небольших единиц функциональности продуктов семейства.

Список литературы

[1] Бабецкий Г.И. и др. АЛЬФА — система автоматизации программирования. — Новосибирск: 1967. — 308 с.

[2] Бульонков М.А., Бабурин Д.Е. HyperCode — открытая система визуализации программ. // Автоматизированный реинжиниринг программ. — СПб.: 2000. — С. 165-183.

[3] Оносовский В.В., Терехов А.Н. Организация работ в проекте RescueWare // Автоматизированный реинжиниринг программ. — СПб.: 2000. — С. 43-63.

[4] Терехов А. А. Языковые преобразования в задачах реинжиниринга программного обеспечения: Дисс. канд. физ.-мат. наук. — СПб.: 2002. — 82 с.

[5] Терехов А.Н., Эрлих Л.А., Терехов А.А. История и архитектура проекта RescueWare // Автоматизированный реинжиниринг программ. — СПб.: 2000. — С. 7-19.

[6] Эрлих Л.А. Технология реинжиниринга и компонентизации устаревших программных комплексов: Дисс. канд. физ.-мат. наук. — СПб.: 2002.

[7] COOL:Enterprise 4.3. ADW/Design Workstation Use Guide. — Stirling Software Inc., 1997.

[8] COOL:Gen product description. — http://www3.ca.com/Solutions/Product.asp

[9] Bass L., Clements P., Kazman R. Software Architecture in Practice. — Boston: Addison-Wesley, 1998. — 560 p.

[10] Jaaksi A. Developing Mobile Browsers in a Product Line // IEEE Software, July/August 2002. — P. 73-80.

[11] Maccari A., Riva C. Architecture Evolution of Legacy Product Families // Proc. of 4th International Workshop on Software Product-Family Engineering. — 2002. — P. 64-69.

[12] Northrop L.M. SEI's Software Product Line Tenets // IEEE Software, July/August 2002. — P. 32-41.

[13] SAA Cross System Product: Designing and Developing Applications. — IBM, 1992.

[14] SEER*HPS 5.4.0. Developing Applications. Edition 1. — SEER Technologies Inc. 1997.

[15] Software Engineering Institute. Capability Maturaty Model. Integrated for System Engineering / Software Engineering. v. 1.0. CMU/SEI-00-TR-018. — Carnegie Mellon University, 2000.

[16] Terekhov A.A., Kiyaev V.I. Software Process Improvement in Russian Company: A Case Study // Proc. of the 1st International Workshop "New Models of Business: Managerial Aspects and Enabling Technology". — 2001. — P. 122-130.