УДК 681.3

МЕТОД ОБРАТНОЙ СЕМАНТИЧЕСКОЙ ТРАССИРОВКИ ДЛЯ КОНТРОЛЯ КАЧЕСТВА В ГИБКОЙ РАЗРАБОТКЕ ПРОГРАММНЫХ ПРОЕКТОВ

В.Л. Павлов, К.А. Жереб, А.Е. Дорошенко, В.И. Сергиенко

Международный НИИ проблем программирования INTSPEI,
03038, Киев, Украина, ул. Николая Гринченка, 4.
Тел.: +380-(95) 874 2832, e-mail: kzhereb@intspei.com

Метод обратной семантической трассировки предназначен для контроля качества программных проектов путем снижения рисков несоответствия требованиям входных и выходных артефактов на каждом этапе процесса разработки. В работе предложена методика применения этого метода при гибкой разработке проектов на примере использования методологии MSF Agile.

The reverse semantic traceability is a method of controlling quality of software project artifacts by decreasing risks of disparity between input and output artifacts on every project stage. The paper discusses the application of this method in agile software development in the case of Microsoft Solutions Framework for Agile.

Введение

С увеличением масштаба и уровня интеллектуализации программного обеспечения (ПО) значительно повышаются также и требования к качеству как самого ПО, так и процессов его производства [1]. Характерно, что в большинстве существующих циклов разработки ПО наиболее важные решения принимаются в начале проекта. Соответственно, ошибки на начальных стадиях проекта оказываются наиболее дорогими, а по мере разработки стоимость исправления ошибки уменьшается. При этом контроль качества минимален в начале проекта и усиливается в процессе разработки. Таким образом, важные ошибки этапа анализа и проектирования могут быть обнаружены только на поздних стадиях разработки, что приводит к дорогостоящей переработке.

Существует два решения этой проблемы: можно либо уменьшать длительность итераций разработки, либо вводить дополнительные средства контроля качества, которые позволят обнаруживать ошибки этапа анализа и проектирования по мере их возникновения, а не на поздних стадиях разработки. Первый подход стал популярным в последние годы, с возникновением гибких (agile) методов разработки [2]. Короткие итерации позволяют получать своевременные отзывы от заказчика; кроме того, средства автоматизированного тестирования можно применять на более ранних стадиях проекта. Но для больших проектов сократить итерации часто просто невозможно; поэтому необходимо использовать другие подходы для контроля качества. Эти подходы представлены различными вариантами инспектирования (software review) [3], а также методом обратной семантической трассировки (Reverse Semantic Traceability), входящим в состав методологии INTSPEI P-Modeling Framework [4].

Методология INTSPEI P-Modeling Framework успешно использовалась в больших проектах; результаты первого применения описаны в [5]. Первые пользователи рекомендовали ее также использовать для гибкой разработки. Для этого был разработан интегрированный процесс, содержащий методологию INTSPEI P-Modeling Framework и один из гибких процессов – Microsoft Solutions Framework for Agile Software Development (MSF Agile) [6]. В работе [7] описан опыт интеграции методологии INTSPEI P-Modeling Framework с MSF Agile и предоставлено техническое описание интегрированного процесса. Данная работа концентрируется на методологических аспектах применения INTSPEI P-Modeling Framework в проектах гибкой разработки. Описан также пример использования обратной семантической трассировки для реализации одной известной задачи на основе методологии гибкой разработки.

1. Обратная семантическая трассировка

В этом разделе приведено краткое описание нового метода контроля качества, обратной семантической трассировки (ОСТ), которая является важнейшей частью методологии INTSPEI P-Modeling Framework. Также мы сравниваем ОСТ с близкими подходами к контролю качества.

1.1. Основы метода

Обратная семантическая трассировка – метод контроля качества, который позволяет проверять соответствие между входными и выходными артефактами каждого шага процесса. Для каждого такого шага, перед переходом к следующему шагу, входные элементы для текущего шага восстанавливаются из выходных элементов. После этого восстановленные версии входных элементов сравниваются с исходными версиями. Если между этими версиями есть семантические различия, текущий шаг процесса повторяют, стараясь устранить несоответствия между входными и выходными артефактами.

Ключевое слово в названии этого метода – "семантическая", потому что оригинальную и восстановленную версии артефакта сравнивают семантически, по "значению" этих артефактов, а не механическим подсчётом терминов. Поскольку сегодня такие операции трудно автоматизируемы, то восстанавливать входные артефакты и оценивать семантические различия должен человек. Такой пересмотр позволяет получить численную оценку трассируемости семантики от входного к выходному артефакту.

Предыдущие работы по методу ОСТ сосредотачивались на идеях метода [4, 8], а также его применении в образовании [9] и больших индустриальных проектах [5, 7]. В частности, в [5] показано, как метод ОСТ успешно работает для больших проектов, которые используют формализованные методологии, такие как RUP [10]. Цель данной работы – сравнительное обсуждение свойств методологии INTSPEI P-Modeling Framework и демонстрация применения системы на конкретном примере.

1.2. Другие близкие системы

Метод ОСТ усиливает существующие способы контроля качества программного обеспечения, как-то инспекции ПО [3], благодаря элементам обратной инженерии и управления трассировкой проектных артефактов. В частности, ОСТ похожа на процедуру инспекционной проверки [3], когда инженер, выполняющий обратное проектирование (в роли рецензента), вычитывает "текст" преобразованного артефакта, чтобы восстановить оригинал артефакта. В работах [11, 12] предложена подобная методика для чтения объектно-ориентированного кода. Однако, особенность ОСТ в том, что рецензенту не позволяют, а запрещают знакомиться со входными артефактами – он должен восстановить их на основе выходных. ОСТ добавляет в процесс рецензирования еще один шаг, а именно, сравнение восстановленного и оригинального артефакта. Это помогает удостовериться, что выходной артефакт не противоречит ранее созданным входным артефактам (например, архитектура соответствует требованиям, или ошибка устранена согласно описанию дефекта). Этот дополнительный шаг также усиливает эффект инспектирования, поскольку рецензент должен исследовать выходной артефакт более тщательно, чтобы восстановить входы.

Имеется также много решений для трассировки зависимостей между проектными артефактами на различных этапах жизненного цикла ПО [13–15]. Трассировка зависимостей используется для установления связей между требованиями и фрагментами исходного кода, реализующего эти требования, а также для отслеживания результатов изменения требований в исходном коде. Исследования в этой области концентрируются на метамоделях процессов трассируемости [14], сценариях использования трассируемости [14, 15] и автоматическом создании связей трассируемости [16]. Однако, большинство решений трассируемости даёт только средства для связывания артефактов, но не для оценки качества этих связей. Метод ОСТ расширяет отношение трассируемости, назначая числовые качественные значения связям трассируемости, например, связи требований с архитектурой проекта.

2. Интеграция с MSF Agile

INTSPEI P-Modeling Framework – это надстройка над существующими методологиями разработки, а не автономный продукт. В настоящее время методология интегрирована как с гибкими, так и формальными процессами. К первой категории относятся Microsoft Solutions Framework for Agile Software Development (MSF Agile) [6], и Open Unified Process [17], а ко второй – IBM Rational Unified Process [10], и Microsoft Solutions Framework for CMMI Process Improvement [6]. В данной работе проиллюстрируем интеграцию методологии с гибкими процессами на примере MSF Agile. Детали технического описания MSF Agile приведены в [6].

2.1. Элементы подхода INTSPEI P-Modeling Framework для встраивания в MSF Agile

Ключевыми элементами методологии P-Modeling Framework являются методы обратной семантической трассировки и безмолвного моделирования [5]. Методология P-Modeling, интегрированная с MSF Agile, содержит детальные описания указанных методов и пошаговые инструкции для членов команды разработчиков. Она также описывает, как обратная семантическая трассировка и безмолвное моделирование работают вместе с другими процедурами MSF Agile.

Методология P-Modeling Framework, интегрированная с MSF Agile, предполагает такие действия (activities) ОСТ:

- выполнить ОСТ для сценария;
- выполнить ОСТ для архитектуры решения;
- выполнить ОСТ для реализации задания на разработку;
- выполнить ОСТ для реализации в базе данных;
- выполнить ОСТ для исправления ошибки;
- выполнить ОСТ для сценариев проверки (test cases);
- выполнить ОСТ для сценариев проверки требований по качеству обслуживания.

Действия ОСТ встроены в процесс MSF Agile. Методология INTSPEI P-Modeling Framework рекомендует выполнить сессию ОСТ при создании важных рабочих продуктов (work product), согласно MSF Agile, чтобы проверить, что никакая информация не была потеряна или извращена в процессе работы. Типичные сессии ОСТ включают следующие шаги: подготовка, реинженерия, экспертная оценка и принятие решения.

На время сессии ОСТ её участникам назначают специальные роли. Методология INTSPEI P-Modeling Framework добавляет к набору ролей MSF Agile три новых роли: "Владелец артефакта" (Artifact Owner), "Реинженер" (Reverse Engineer), "Эксперт", и использует одну из ролей MSF, – «Менеджер проекта». Роли ОСТ сочетаются с ролями MSF: например, лицо, выполняющее роль "Архитектор" в MSF Agile, получит роль ОСТ "Владелец артефакта" на время сессии ОСТ для Архитектуры решения (Solution Architecture). Одно и то же лицо может выполнять различные роли в различных сессиях ОСТ; более того, эта практика поощряется, чтобы команда разработчиков быстрее освоилась с методологией INTSPEI P-Modeling Framework.

Результаты сессии ОСТ оформляются в виде специальных рабочих продуктов – “Отчетов сессий ОСТ” (RST Session Report) и “Оценок экспертов” (RST Expert Assessment). Отчет сессии ОСТ содержит всю информацию о сессии, включая дату проведения, состав участников, оригинальные и восстановленные артефакты и окончательное решение. Оценка экспертов фиксирует результат обсуждения экспертами и их оценку качества шага разработки вместе с их комментариями. Эти рабочие продукты используются для сообщения результатов сессии ОСТ команде разработчиков.

Основываясь на результатах сессии ОСТ, менеджер проекта принимает одно из следующих решений:

- качество артефактов достаточно, и разработка может переходить к следующей стадии;
- необходима доработка выходных артефактов, чтобы устранить дефекты и потерю информации;
- необходимо исправить входные и выходные артефакты для устранения недоразумений;
- требуется доработать артефакты и провести еще одну сессию ОСТ.

Если обратную семантическую трассировку применять ко всем рабочим продуктам, это приведет к существенным дополнительным затратам. Поэтому INTSPEI P-Modeling Framework рекомендует ранжировать рабочие продукты по приоритетам и выполнять сессии ОСТ только для наиболее важных из них. Приоритеты должен расставить менеджер проекта во время планирования итерации (в начальной стадии каждой итерации). Артефакты ранжируют по нескольким критериям, включая их влияние на качество конечного продукта, серьезность возможных дефектов и возможность использования других методов контроля качества. Результаты этого подсчёта записывают в "Таблицу ранжирования ОСТ" (RST Rank Table).

3. Практический пример

В этой разделе описано применение методики ОСТ на примере небольшого проекта, выполненного согласно гибкой методологии MSF Agile для программной реализации известной задачи – игры "Жизнь" [18].

3.1. Описание проекта

Проект разрабатывался два месяца силами трёх разработчиков-студентов, работавших по совместительству.

Проектная команда придерживалась процесса MSF Agile, согласно описанию MSF Process Guidance версии 4.1 [6]. Проект состоял из четырёх двухнедельных итераций. Первая итерация отвечала только трекам "Выработки концепции " (Envision) и "Планирование". Команда создала общее описание проекта, собрала требования и составила первоначальный дизайн. Две промежуточные итерации ушли на собственно разработку и тестирование продукта (трек "Создание", Build). Последняя итерация сочетала треки "Разработка" и "Стабилизация".

Команда разработчиков использовала методологию P-Modeling Framework с самого начала проекта. В первый день первой итерации распланировали сессии ОСТ. Для каждой сессия потребовался минимум один участник, не знакомый с подробностями проекта, так что менеджеру проекта следовало найти их заранее. Таким образом, менеджер проекта должен был представлять себе, сколько сессий ОСТ надо провести и для каких артефактов.

В нашем проекте команда заметила, что сессии ОСТ на первой и последующих итерациях различаются. На первой итерации важнейшими документами являются "Описание проекта" (Vision) и "Дизайн верхнего уровня". Если на этом этапе для них проделать обратную семантическую трассировку, то это заметно помогает процессу MSF. Качество таких артефактов не удаётся эффективно проверить другими способами, по крайней мере, пока не начнётся реализация на последующих итерациях. И напротив, на последующих итерациях ОСТ смещается на более "мелкие" артефакты: реализацию задач разработки, исправления ошибок и тестирование. На этих итерациях обратная семантическая трассировка только дополняет другие методы контроля качества, как-то модульные и функциональные тесты.

Методология P-Modeling Framework для MSF Agile предполагает верифицировать описание проекта, восстанавливая его из сценариев – этот процесс описан в операции (Activity) "ОСТ для Сценариев"; дизайн верхнего уровня проверяют, восстанавливая из него требования. Планируя ОСТ, команда решила, что проверить дизайн гораздо более важно. Описание проекта было достаточно простым, и команда не ожидала в нём существенных дефектов. Оказалось, что дизайн содержал существенные дефекты, потому что у создавшего его студента не было достаточного опыта разработки программ. Разработчики могли утерять информацию, переходя от требований к дизайну, поскольку требования и дизайн разрабатывали разные люди. Ниже для иллюстрации рассмотрены только действия (activities) обратной семантической трассировки из первой итерации.

3.2. Сессия ОСТ для дизайна

Сессия ОСТ для дизайна была важнейшей из обратных семантических трассировок проекта. На первой итерации команда создала дизайн; на сессии ОСТ реинженеры (reverse engineers), не знакомые ранее с проектом, восстановили требования. После этого команда экспертов сравнила восстановленную и оригинальную версии требований и дала замечания.

Первоначальные требования были разделены на функциональные и нефункциональные, или, согласно MSF Agile, истории (scenario) и требования качества (quality of service). Требования хранились в системе Microsoft Team Foundation Server. На рис. 1 показаны требования, выявленные аналитиком.

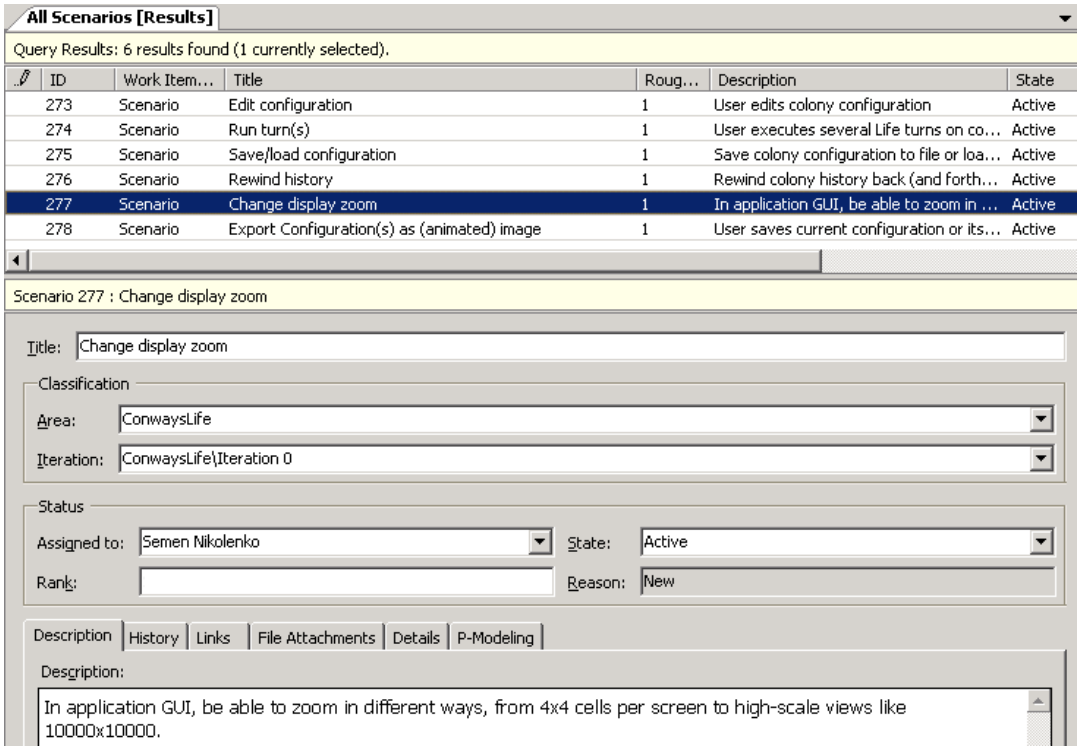

Рис.1. Первоначальные сценарии проекта

Главными сценариями являются: редактирование конфигурации, прогон моделирования для одного или большего количества ходов и сохранение или загрузка игровой конфигурации. Дополнительные сценарии с более низким приоритетом включают сценарии перемещения вдоль хронологии моделирования, изменения размера отображаемой конфигурации и экспортирования конфигурации как изображения (полный текст требований не включен в текст статьи из-за ограничений размера).

Основываясь на требованиях, архитектор разработал дизайн приложения, используя инструментарий Microsoft Visual Studio. Основные классы приложения представлены на рис. 2.

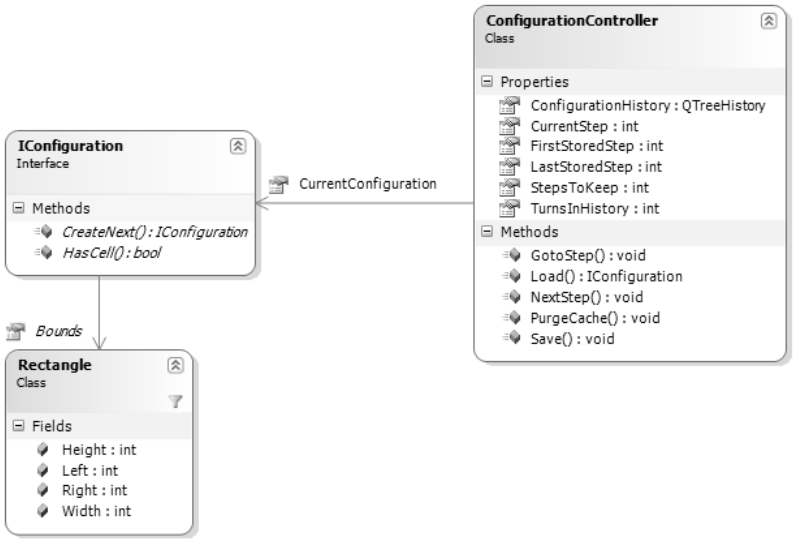

Рис. 2. Основные классы приложения

Интерфейс IConfiguration представляет конфигурацию одиночной колонии в игре; ConfigurationController вычисляет следующий игровой шаг, запоминает историю развития колонии и сохраняет или загружает состояние игры. Классы, ответственные за графический интерфейс пользователя (ConfigurationDisplay и LifePlayer) показаны на рис. 3. Рис. 4 отображает классы, связанные с алгоритмом "Hashlife" [18], который выполняет моделирование.

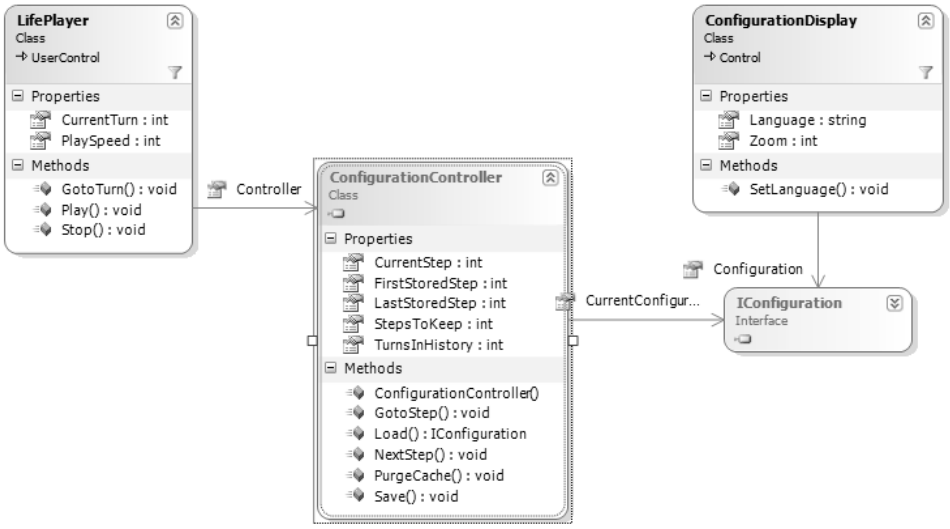

Рис. 3. Классы графического интерфейса

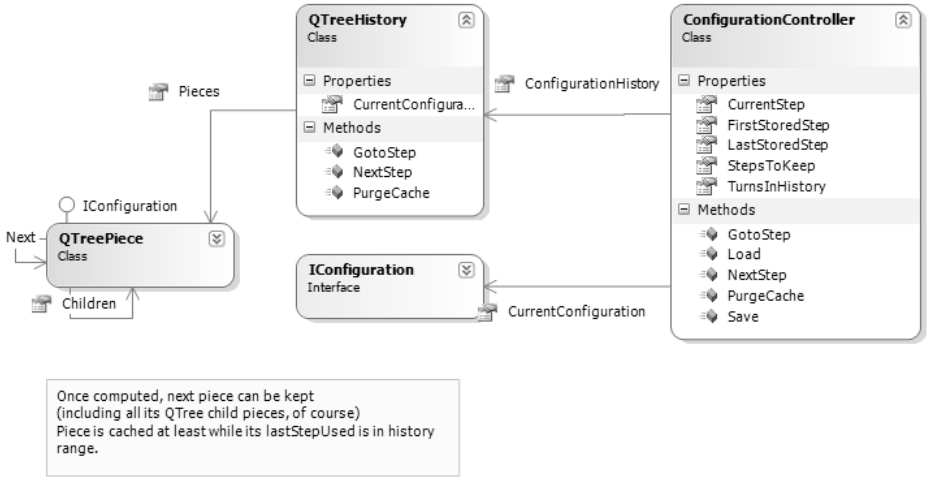

Рис. 4. Классы алгоритма "Hashlife".

Когда диаграммы, представляющие архитектуру, были готовы, был выполнен сеанс OCT, чтобы проверить, что архитектура отвечает требованиям. Реинженеры восстановили требования, не имея никаких предварительных знаний об этом проекте; однако, они знали правила игры «Жизнь». Продолжительность сеанса OCT была ограничена 1 часом. Реинженеры восстанавливали следующие требования (рис. 5).

Сценарий 1: Пользователь может управлять игрой “Жизнь”, выполняя следующие действия:
- Начать игру с помощью команды “Play”
- Остановить игру с помощью команды “Stop”
- Вернуться к заданному ходу с помощью команды “Go to Turn”

Дополнительные требования
- Система моделирует каждый шаг игры со скоростью, определяемой параметром “PlaySpeed:int”
- Система запоминает историю шагов игры
- Игра использует алгоритм «Hashlife» (http://en.wikipedia.org/wiki/Hashlife). Предполагается наличие быстрой памяти для моделирования историй.
- Возможность стереть историю игры не предполагается.

Сценарий 2: Игрок может изменять конфигурацию интерфейса игры. Игрок может задавать язык интерфейса игры.

Дополнительные требования
- Установки интерфейса: язык, масштаб

Примечания:
- Пользовательский интерфейс не позволяет изменять уровень масштаба и скорость моделирования.
- В классе Rectangle ошибка: он должен содержать атрибуты высот и ширины, а также координаты левой верхней вершины.

Рис. 5. Восстановленные требования

После восстановления требования эксперты сравнили их с оригинальными требованиями. Этот процесс тоже занял около часа времени: из них 40 минут ушло на ознакомление экспертов с документами, и 20 минут – на обсуждение. Эксперты пришли к следующим выводам (рис. 6).

1. Требование не восстановлено: редактирование конфигурации.
2. Требование не восстановлено: сохранение/загрузка конфигурации.
3. Требование не восстановлено: обратная прокрутка истории игры.
4. Требование не восстановлено: экспортирование конфигураций как анимированных образов.
5. Восстановлено дополнительное требование: изменение языка.
6. Восстановлено дополнительное требование: очистка истории игры.
7. Пользовательский интерфейс не позволяет изменять язык.
8. Примечание: что такое “конфигурация”? Напоминает “конфигурацию программы” и может быть легко воспринята именно так. Предложены названия “колония” или “доска”.

Рис. 6. Комментарии экспертов

Исходя из результатов сессии OCT, команда приняла следующие решения для доработки проекта:
- изменить требования – добавить недостающий сценарий ("Выбрать язык");
- прояснить словарь предметной области – использовать термин "колония" вместо неоднозначного "конфигурация";
- расширить функциональность "Редактирования конфигурации", что потребует значительного изменения архитектуры;
- более детально проработать классы графического интерфейса пользователя, что потребует незначительных изменений архитектуры.

На рис. 7 показана новая диаграмма классов, которая была создана, в частности, в результате уточнений, предложенных сессией OCT. Она содержит классы, связанные со сценарием, который был первоначально опущен в проекте, и поэтому не восстановлен реинженерами ("Редактирование конфигурации"). Заметим, что другие диаграммы также подверглись изменениям (на рис. 7 не показаны).

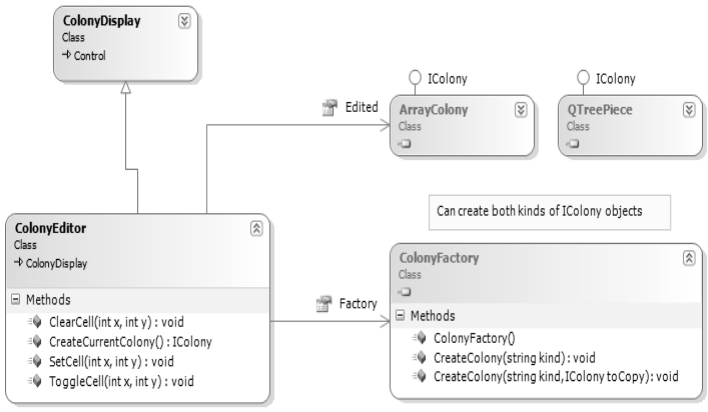

Рис. 7. Изменения в проекте после сессии OCT: новая диграмма

3.3. Анализ

Вышеприведенный пример сессии OCT, выполненной в рамках подхода MSF Agile, продемонстрировал, что обратная семантическая трассировка может быть ценным методом контроля качества в проектах, основанных на гибких методологиях. Наиболее важным способом использования OCT является контроль качества в начальной стадии проекта, на первых итерациях. Даже в случае гибких процессов имеется промежуток времени в начале выполнения проекта, когда никакого программного кода еще нет, и команда сосредотачивается на разработке концепции проекта и важных архитектурных решениях. Типичные методы контроля качества в таких случаях – автоматизированное тестирование модулей и получение отклика от заказчика – имеют весьма ограниченный эффект. Однако метод OCT дает возможность проверять, правильность первых важные артефактов или, по крайней мере, их непротиворечивость.

Члены команды разработчиков были весьма удовлетворены возможностью обнаружения дефектов в артефактах программных проектов без фактического испытания программного обеспечения (и даже написания кода). Единственный метод, который они знали и который мог бы дать сопоставимые результаты, был метод инспекции. Однако метод OCT оказался более эффективным как в отношении затрачиваемых ресурсов, так и качества результата. Полная сессия OCT потребовала около 7 человеко-часов, из них: 1 час на подготовку, 1 час для работы двух реинженеров и 1 час – для четверых экспертов (большая часть этого времени была потрачена людьми, не включенными непосредственно в проект). Однако, результат сессии был весьма ощутимым. И требования, и дизайн проекта были существенно улучшены, а некоторые дефекты были устранены, что избавило разработчиков от значительной переработки проекта, если бы ошибки были обнаружены на этапе фактического кодирования программ. Например, сценарий "Редактирования колонии (конфигурации)", который был опущен при проектировании, потребовал существенных изменений в интерфейсе пользователя, также как в основных классах проекта. Когда уточнения, предложенные OCT, были сделаны, разработчики заметили, что одно из важных предположений относительно классов ядра было неправильным. А именно, класс QTreePiece (см. рис. 4) рассматривался как неизменный (immutable) в соответствии с требованиями алгоритма "Hashlife". Однако, сценарий "Редактирования колонии" требовал, чтобы колонии (реализованные классом QTreePiece) могли изменяться. В результате, разработчики решили добавить еще один, изменяемый, класс, представляющий колонию - ArrayColony. Это изменение в проекте заняло приблизительно 2 часа. Однако, потенциальная стоимость доработки была бы намного больше, если бы неправильное решение было воплощено в коде. Разработчики оценили, что этот дефект (опущенная возможность редактирования), конечно, был бы обнаружен, но его исправление потребовало бы существенных изменений в интерфейсе пользователя, классах ядра и блоках тестирования и могло бы занять нескольких рабочих дней.

Участники проекта и сессии OCT были едины во мнении, что эффект от обратной семантической трассировки не ограничивается улучшением качества артефактов, которые использовались в сессии OCT. Члены команды осознали и усвоили, что проектные артефакты не должны создаваться только потому, что это предусмотрено процессом. Артефакты, созданные одним членом команды, затем будут использоваться другим, и автор артефакта должен сделать артефакт понятным его потребителю. Поэтому, сессии OCT гарантировали, что создаваемые артефакты могли фактически использоваться в проекте, что вполне соответствует принципам гибких методологий, согласно которым артефактов, которые не используются в проекте, нужно избегать. Метод OCT помог превратить проектные артефакты, включая требования и дизайн, в фактически использованные, а не только написанные ради бюрократической процедуры документы.

Заключение

В работе описан метод обратной семантической трассировки для контроля качества при гибкой разработке программного обеспечения, и показано его применение на примере экспериментального проекта. Анализ этого примера продемонстрировал, что обратная семантическая трассировка может использоваться в гибких методологиях разработки и существенно дополнять используемые там методы контроля качества.

Согласно нашим наблюдениям, использование ОСТ наиболее важно в течение первой итерации гибкого проекта.

Будущие направления исследований включают применение методологии INTSPEI P-Modeling Framework в промышленных проектах, основанных на гибких процессах разработки. Также предполагается провести большее количество экспериментов по применению обратной семантической трассировки к различным видам артефактов, улучшить процессы ОСТ, и разработать более детальные инструкции использования методологии INTSPEI P-Modeling Framework как для гибких, так и формальных процессов разработки программного обеспечения.

1. *Андон Ф.И., Коваль Г.И., Коротун Т.И., Лаврищева Е.М., Суслов В.Ю.* Основы инженерии качества программных систем. – 2-е изд., перераб. и доп. – Киев.: Академпериодика, 2007. – 672 с.
2. *Manifesto* for Agile Software Development http://www.agilemanifesto.org
3. *Fagan M.E.*, Design and Code Inspections to Reduce Errors in Program Development. IBM Syst. J. –1976. – Vol. 15, N. 3 – P. 181–211.
4. *Pavlov V. L., Busygin S., Boyko N., Babich A.*, "Is There Still a Room For Programmers' Productivity Improvement?", Proceedings of the 5th East- West Design and Test Symposium (EWDTS'07) – 2007. – P. 146–151.
5. *Pavlov V., Boyko N., Babich A.*, "First Experience of Using INTSPEI P Modeling Framework in Software Development Projects", Problems in Programming, Issue 2 – May 2007 – P. 68–75.
6. *Microsoft* Solutions Framework http://www.microsoft.com/msf
7. *Pavlov V., Doroshenko A., Taganskaya T., Zhereb K., Boyko N*, An Experience of Integrating INTSPEI P-Modeling Framework with Microsoft Solutions Framework for Agile Software Development, 2007 (accepted for publication in Proc. IASTED Int. Conf. Software Engineering, 2008).
8. *INTSPEI* P-Modeling Framework Whitepaper, INTSPEI, http://www.intspei.com
9. *Pavlov V., Yatsenko A.* "'The Babel Experiment': An Advanced Pantomime-based Training in OOA&OOD with UML", 36th 'ACM Technical Symposium on Computer Science Education', February 25, 2005.
10. *Kruchten, Ph.* The Rational Unified Process: An Introduction. Addison-Wesley, 2003.
11. *Dunsmore A., Roper M., and Wood M.*, "Systematic Object-Oriented Inspection – An Empirical Study," Proc. 23rd Int'l Conf. Software Eng. (ICSE '01) – May 2001 – P. 135–144.
12. *Wood M., Roper M., Brooks A., and Miller J*. Comparing and combining software defect detection techniques: a replicated empirical study. SIGSOFT Softw. Eng. – Nov. 1997 N 22, 6. – P. 262–277.
13. *Aizenbud-Reshef N., Nolan B.T., Rubin J., Shaham-Gafni Y.*, Model traceability. IBM SYSTEMS J. – 2006. – Vol. 45, N. 3.
14. *Ramesh B., Jarke M.*, ''Toward Reference Models for Requirements Traceability,'' IEEE Transactions on Software Engineering 27, N. 1 – January 2001 – P. 58–93.
15. *Gotel O.C.Z., Finkelstein A.C.W.*, ''An Analysis of the Requirements Traceability Problem,'' Proceedings of the First International Conference on Requirements Engineering, Utrecht, The Netherlands. – 1994. – P. 94–101.
16. *Antoniol G., Canfora G., Casazza G., Lucia A.D., and Merio E.* Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 28(10), October 2002.
17. *Kroll P., MacIsaac B.* Agility and Discipline Made Easy – Practices from OpenUP and RUP, Addison-Wesley Professional, 2006. – 448 p.
18. *Conway's* Life Game, http://en.wikipedia.org/wiki/Conway's_Game_of_Life