УДК 004.414.3

КОНЦЕПЦИЯ УПРАВЛЕНИЯ ТРЕБОВАНИЯМИ К ПРОГРАММНЫМ СРЕДСТВАМ

В.В. БАХТИЗИН, С.Н. НЕБОРСКИЙ

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 24 января 2006

В данной статье рассмотрена концепция управления требованиями к программным средствам, а также предложен подход к ее реализации. Предложенный подход позволяет установить связь между объектами программной реализации некоторого требования и данным требованием. Подход основан на системе управления требованиями Telelogic DOORS, и применим к программным средствам, которые разрабатываются для функционирования в среде .NET.

Ключевые слова: управление требованиями, требования к программным средствам.

Введение

Прежде чем начинается разработка любого программного средства (ПС), определяются требования, которые предъявляются к нему. От того, насколько правильно, корректно и точно сформулированы требования, непосредственно зависит качество конечного продукта. Требования – это сформулированные и представленные в некотором виде описания возможностей, которые должно предоставлять ПС, а также ограничения, относящееся к ПС. Можно рассматривать требования как свойства, которыми должен обладать продукт, чтобы представлять какую-то ценность для пользователей [1]. Очевидно, чем сложнее разрабатываемое ПС, тем более сложную структуру приобретают сами требования, а значит, сами требования нуждаются в определенных ПС для работы с ними. Такие ПС называют системами управления требованиями (СУТ).

Управление требованиями — это процесс поддержания требований в таком состоянии, при котором обеспечивается подчинение их единой структуре и четкая организация взаимосвязи между ними. Результат использования СУТ — выигрыш во времени разработки ПС, в надежности ПС, и, самое главное, в том, что полученный продукт будет именно таким, каким его хотел видеть заказчик. Время разработки ПС снижается за счет сокращения документооборота между разработчиком и заказчиком. Качество полученного ПС повышается, так как СУТ позволяет прослеживать связь от исходного требования пользователя до результата тестирования этого требования.

Одной из наиболее распространенных СУТ в мире является Telelogic DOORS. В качестве примеров других СУТ можно привести IBM Rational RequisitePro, Borland CaliberRM. Следует отметить, что DOORS имеет ряд преимуществ перед приведенными системами, поэтому метод реализации требований, предлагаемый в данной работе, опирается на механизм управления требованиями DOORS. Telelogic DOORS предоставляет следующие возможности:

формулирование самих требований;

определение всей необходимой для управления требованиями информации;

контроль состояния требований;

гибкая политика изменения требований;

проверка соответствия всей информации, описывающей требования, установленным стандартам и правилам;

обеспечение полного анализа влияния изменения требований.

Концепция управления требованиями

В СУТ DOORS требования описываются в иерархическом виде. Причем собственно требования заключены в так называемые формальные модули. В каждом формальном модуле требования представлены в виде древовидной структуры, где листьями дерева являются непосредственно требования, а ветвями — обобщающие их секции.

Целью иерархического представления требований является их структуризация. Структуризация ускоряет как процесс разработки требований, так и процесс их уяснения теми, кто будет реализовывать данные требования. Однако для управления требованиями иерархического представления недостаточно. Для управления в DOORS реализован механизм трассировки и введено понятие связи.

Связи позволяют эффективно управлять изменениями. Благодаря им можно быстро проследить влияние изменения одного отдельного требования на всю систему. Невозможно управлять требованиями эффективно без трассируемости требований. Требование является трассируемым, если можно определить, кто предложил его, почему это требование существует, какие требования связаны с данным требованием и как данное требование связано с другой информацией, такой, как архитектура системы, ее реализация и пользовательская документация [2]. Трассируемость требований документирует зависимости и логические связи отдельных требований и других элементов системы. К этим элементам относятся требования различных типов, бизнес-правила, архитектура и другие компоненты дизайна, модули исходного кода, варианты тестирования и файлы справки [1].

Формально управление требованиями можно описать в виде множества требований R и матрицы связей, или матрицы трассируемости, M. Множество требований может быть представлено следующим образом:

$$R = \left\{ r_i \;\middle|\; i = \overline{1,n} \right\},$$

где n — количество требований; r_i — i-е требование.

Чтобы реализовать трассируемость, каждое требование должно быть уникально и последовательно идентифицировано [1]. Таким образом, в качестве r_i могут выступать идентификаторы требований. Матрица связей между требованиями может быть представлена как квадратная матрица размерностью $n \times n$, строки и столбцы которой соответствуют элементам множества R:

$$M = \left\| a_{ij} \right\|_{n \times n},$$

где $a_{ij} = \begin{cases} 1, \text{ если } r_j \text{ зависит от } r_i; \\ 0, \text{ если } r_j \text{ не зависит от } r_i. \end{cases}$

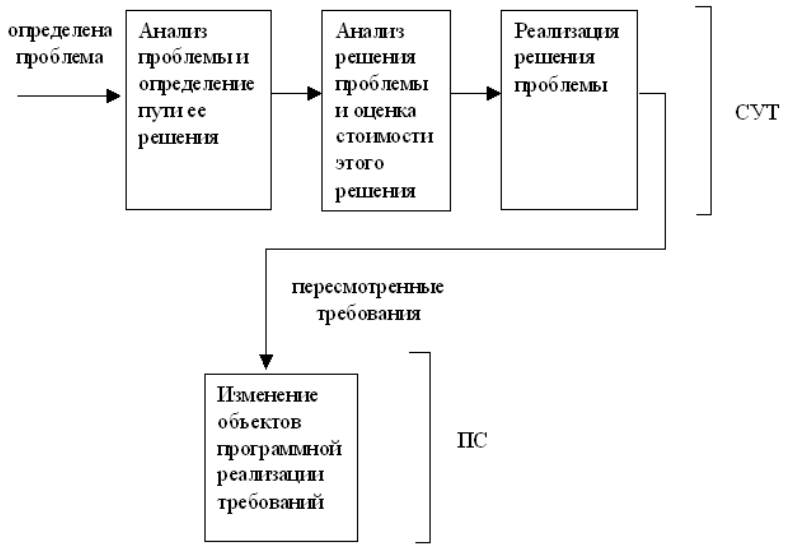
Следует отметить, что все связи между требованиями — однонаправленные. Это означает, что если требование r_j зависит от требования r_i, нельзя утверждать, что r_i зависит от r_j. В качестве примера можно привести следующую матрицу трассируемости, в которой требование r_3 зависит от r_2, а r_2 зависит от r_1:

Пример матрицы трассируемости

	r_1	r_2	r_3
r_1	0	1	0
r_2	0	0	1
r_3	0	0	0

Реализация концепции управления требованиями

В предельно упрощенном виде концепцию управления требованиями можно представить так: при изменении одного требования в рамках системы DOORS можно сразу определить и те требования, на которые данное изменение влияет. Очевидно, такой подход позволяет оперировать самими требованиями, корректировать их. И если произошла модификация еще не реализованных требований, то, естественно, данное ПС не претерпевает никаких изменений. Однако если происходит исправление уже реализованного требования, необходим некоторый механизм для выявления классов, методов, функций программной реализации этого требования. Другими словами, необходим такой метод, который позволит выявить определенные фрагменты программной реализации требования при изменении его. Этапы процесса изменения требования приведены на рисунке.



Процесс изменения требований

Следует отметить, что проблема определения связи между требованием и его реализацией не нова. Существуют различные способы, которые позволяют ее разрешить. Например, можно при определении самих требований ввести дополнительные атрибуты, описывающие объекты программной реализации, и при изменении требования сразу определять, на что именно в программной реализации влияет данное изменение. Однако создание требований и их реализация должны быть, в идеале, разными процессами [2]. Поэтому идея хранения информации о требовании и об объектах программной реализации в одном месте, доступном в равной степени и заказчику, и разработчику, не является приемлемой. Более того, этот способ не является автоматизированным, поэтому не защищен от ошибок. Несколько автоматизировать этот процесс можно, используя язык DXL (DOORS eXtension Language), который расширяет возможности системы DOORS. Однако и здесь не исключены ошибки.

Очевидно, для того чтобы можно было установить корректную связь от требования к его реализации, необходимо искать решение вне рамок СУТ DOORS. Здесь следует отметить использование внешних ПС, т.е. ПС, не относящихся непосредственно к системе DOORS.

Если говорить об автоматизированных способах, то в первую очередь следует описать такой программный продукт как Telelogic Synergy. Этот продукт ориентирован на управление

изменениями и конфигурацией. Интеграция Synergy и DOORS дает трассируемость от требований к объектам исходного кода, также она позволяет анализировать влияния требования или изменения требования на исходный код.

Также стоит отметить линейку программных продуктов SCADE (Safety-Critical Application Development Environment). SCADE DOORS Link предоставляет возможность проследить связь от требований к ПС, описанных в DOORS, к самому программному коду. Однако для того, чтобы воспользоваться возможностями SCADE DOORS Link, необходимо строить весь процесс разработки ПС на основе среды SCADE, что не всегда является приемлемым или возможным.

В силу того что использование внешних ПС, позволяющих проследить связь от требования к его реализации, требует дополнительных усилий по настройке и администрированию этих ПС или просто неприемлемо по каким-либо причинам, стоит задуматься о том, чтобы хранить эту связь в самом программном коде реализации требований. Самый простой способ — хранение подобной информации в текстовом виде путем включения ее в комментарии. Несмотря на то что такой подход используется, он не является автоматизированным. А попытка его автоматизации как раз и приводит к созданию внешних вспомогательных ПС.

В данной работе предлагается способ реализации концепции управления требованиями к ПС, которые разрабатываются для функционирования в среде .NET. Предлагаемый способ позволит решать проблему локализации кода реализации требования автоматически, не прибегая к использованию каких бы то ни было внешних ПС. Он заключается в том, что каждое требование (или набор требований) реализуется в виде одной отдельной сборки. Сборка .NET – это коллекция файлов, которая предстает перед пользователем в виде библиотеки динамической компоновки или исполняемого файла. Сборки являются базовыми единицами платформы .NET для повторного использования кода, контроля версий, защиты и развертывания [3].

Для того чтобы одна сборка могла взаимодействовать с другой, она должна содержать ссылку на эту сборку. Тогда для реализации концепции управления требованиями те сборки, которые реализуют зависимые требования, должны быть также зависимы. Тогда при внесении изменения в программную реализацию одного требования все зависящие сборки будут определены автоматически.

Так как управление версиями встроено в саму платформу .NET [4], то можно говорить о гибкости такого подхода. Так, можно вести разработку новой версии сборки и продолжать использовать старую сборку для совместимости. Или, если по каким-то причинам будет необходимо восстановить использование старой сборки, достаточно просто установить корректную ссылку на нее. Таким образом, предлагаемый метод хорошо применим при построении процесса разработки ПС на основе спиральной модели жизненного цикла.

Формально описать предлагаемый подход можно при помощи множества коллекций сборок A:

$$A = \left\{ C_i \;\middle|\; i = \overline{1,k} \right\},$$

где k — количество коллекций сборок проекта; $C_i = \left\{ V_{ij} \;\middle|\; j = \overline{1,p} \right\}$ — i-я коллекия сборок разных версий, каждая из которых реализует i-е требование (группу требований); p — количество версий сборок, реализующих i-е требование; V_{ij} — i-я сборка (сборка, реализующая i-е требование или группу требований) j-й версии.

Преимуществом описываемого подхода к реализации требований является то, что программные реализации требований могут разрабатываться на разных языках программирования. Так, .NET поддерживает многоязыковую разработку [4], где каждая сборка может быть реализована на любом .NET-совместимом языке (например, на Microsoft Visual C# .NET или Microsoft Visual Basic .NET).

На первый взгляд, может показаться, что при использовании данного подхода значительных усилий требует корректная расстановка ссылок, модификация ссылок при добавлении новой сборки (нового зависимого требования), установление правильного порядка компилирования сборок и т.д. Однако среда Microsoft Visual Studio .NET, являющаяся одной из самых по-

пулярных интегрированных сред разработки .NET приложений на сегодняшний день, позволяет значительно упростить этот процесс.

Для того чтобы интегрированная среда разработки применяла различные инструменты, средства создания графического интерфейса, шаблоны, настройки, Visual Studio .NET реализует концептуальные контейнеры, называемые решениями и проектами. Проект включает набор файлов с исходным кодом, а также различную метаинформацию, такую как ссылки на компоненты и инструкции построения проекта. После построения проекта создается один или более выходных файлов, которые и являются сборкой. Решения включают один или более проектов, а также дополнительные файлы и метаинформацию, которые позволяют определить решение как целое.

Современный процесс разработки ПС немыслим без автоматической генерации кода. И здесь следует отметить, что предложенный подход также может быть автоматизирован. Подобно тому, как код классов и методов может генерироваться на основе метаязыков или диаграмм, используя сами требования и связи между ними, можно автоматически генерировать код решения.

Заключение

Предложенный метод реализации концепции управления требованиями к ПС может быть применен при выполнении таких работ процесса разработки ПС, как проектирование программной архитектуры и техническое проектирование ПС [5]. Причем, как было замечено, этап проектирования может быть автоматизирован на основе информации о требованиях и связях между ними.

Используя данный метод на практике, разработчик не тратит время и усилия на конфигурирование и администрирование связующих ПС, т.е. таких ПС, которые призваны установить связь от определения требований к их реализации и наоборот. Более того, благодаря политике версий сборок, разработчик получает дополнительную гибкость при реализации изменяющихся требований. Таким образом, позволяя локализовать код реализации требования автоматически, метод создает определенные предпосылки для улучшения качества ПС, а также ускоряет весь процесс разработки ПС в условиях изменяющихся требований к нему.

THE CONCEPT OF SOFTWARE REQUIREMENTS MANAGEMENT

V.V. BAKHTIZIN, S.N. NIABORSKI

Abstract

This article describes the concept of software requirements management and an approach to its implementation. The approach's main goal is to provide a link between source code objects implementing some software requirement and this requirement. The approach is based on Telelogic DOORS system, a powerful tool that helps to capture, track and manage requirement. The approach is going to be applied to .NET applications.

Литература

1. *Вигерс К.* Разработка требований к программному обеспечению. М., 2004.
2. *Kotonya G., Sommerville I.* Requirements Engineering. Processes and techniques. John Wiley & Sons, 1998.
3. *Либерти Д.* Программирование на C#. СПб., 2003.
4. *Stiefel M., Oberg R.* Application development using C# and .NET. Prentice Hall PTR, 2002.
5. СТБ ИСО/МЭК 12207-2003: Информационные технологии. Процессы жизненного цикла программных средств.