Программная инженерия

Программные требования
(Software Requirements)

Глава базируется на IEEE Guide to the Software Engineering Body of Knowledge[(1)] - SWEBOK®, 2004. Содержит перевод описания области знаний SWEBOK® “Software Requirements”, с комментариями и замечаниями[(2)].

Сергей Орлик, Юрий Булуй.

Программные требования (Software Requirements)

Программные требования – Software Requirements – свойства, которые должны быть надлежащим образом представлены для решения конкретных практических задач. Данная область знаний касается вопросов извлечения (сбора), анализа, специфицирования и утверждения требований.

Опыт индустрии информационных технологий однозначно показывает, что вопросы, связанные с управлением требованиями, оказывают критически-важное влияние на программные проекты, в определенной степени - на сам факт возможности успешного завершения проектов. Только систематичная работа с требованиями позволяет корректным образом обеспечить моделирование задач реального мира и формулирование необходимых приемочных тестов для того, чтобы убедиться в соответствии создаваемых программных систем критериям, заданным реальными практическими потребностями.

В то же время, возможен, и часто используется на практике и в различных методологиях разработки ПО, альтернативный подход, базирующийся на определении групп требований *к продукту*. Такой альтернативный подход обычно включает группы (типы, категории) требований, например: системные, программные, функциональные, нефункциональные (в частности, атрибуты качества) и

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

т.п. Классический пример (см. рисунок 1) высокоуровневого структурирования групп требований как требований к продукту описан в работах одного из классиков дисциплины управления требованиями – Карла Вигерса.

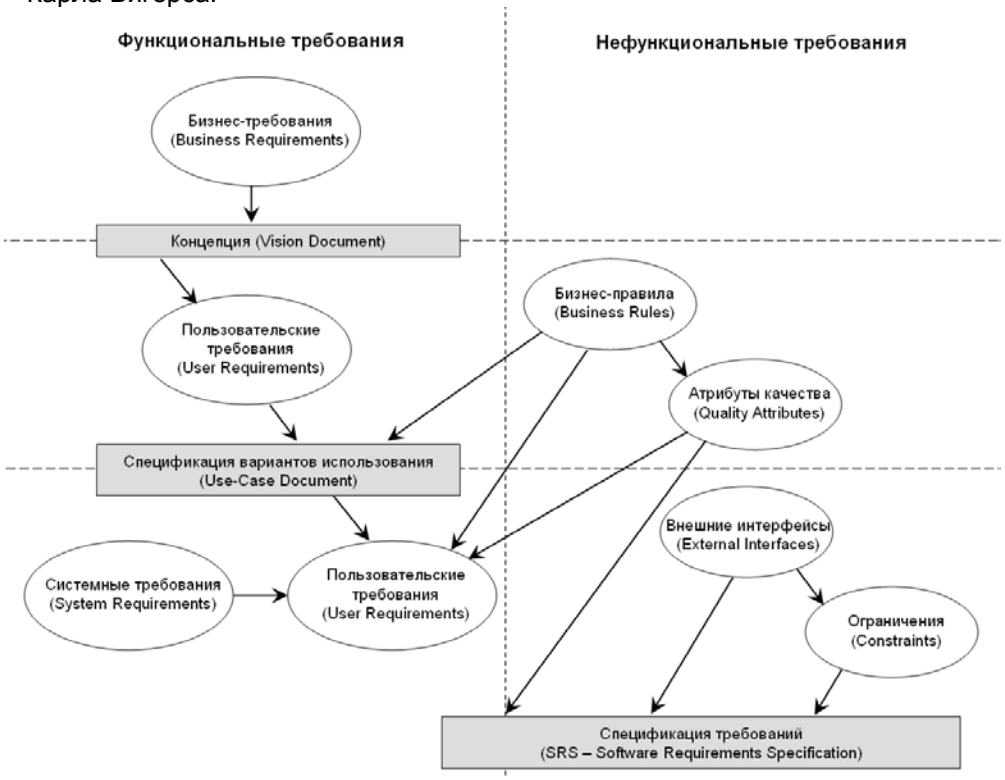

Рисунок 1. Уровни требований по Вигерсу [Вигерс, 2003, с.8, рис. 1-1]

Если с “продуктовой” точкой зрения, интуитивно, более менее все ясно (или, как минимум, знакомо), подход SWEBOK в отношении требований требует детализации.

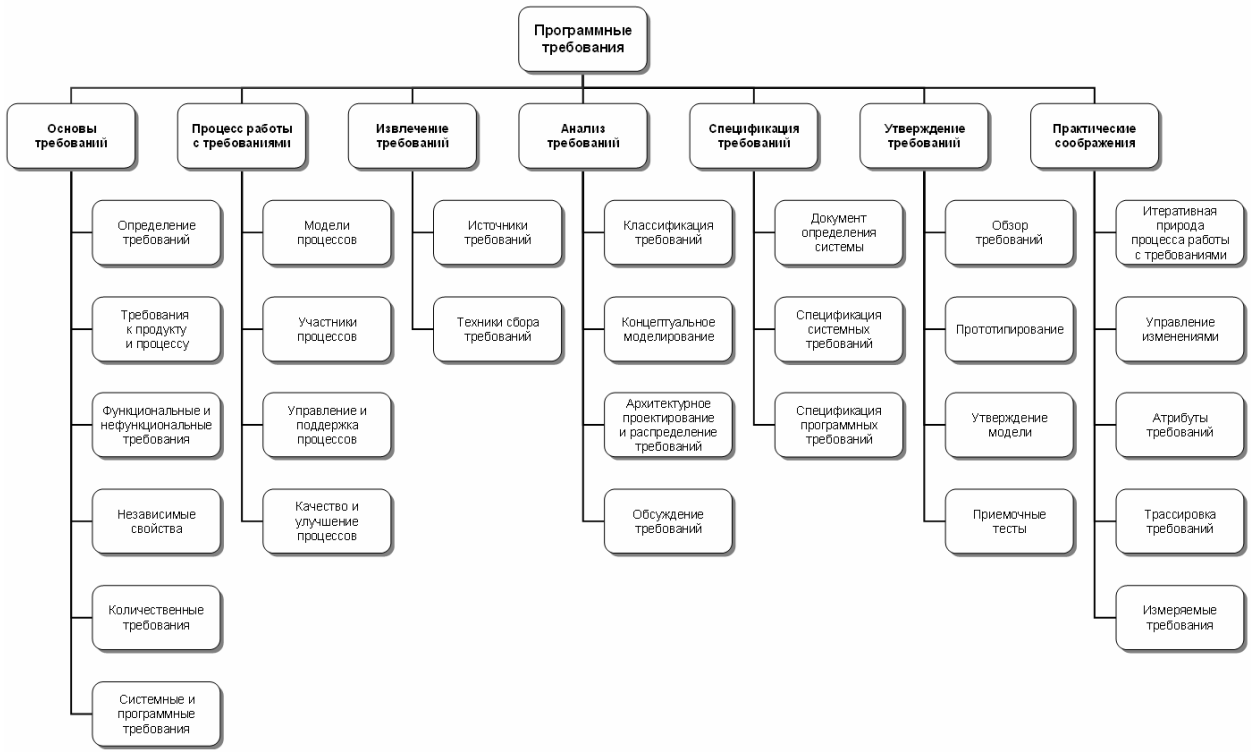

Рисунок 2. Область знаний “Программные требования” [SWEBOK, 2004, с.2-2, рис. 1]

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

Сама же структура обсуждаемой области знаний в большой степени совместима со стандартами IEEE 12207.x, ISO/IEC, ГОСТ Р ИСО/МЭК 12207 (структура стандарта будет рассмотрена позднее). Такая структура построена исходя из идеи выделения ключевых групп вопросов дисциплины.

Область знаний управления требованиями включает 7 секций, каждая из которых представлена в виде ключевых тем (см. рисунок 2). Кроме того, данная область знаний тесно связана со следующими областями:

- Software Design
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Quality

1. Основы программных требований (Software Requirements Fundamentals)

Эта секция включает определение программных требований как таковых и описывает основные типы требований и их отличия: к продукту и процессу, функциональные и нефункциональные требования и т.п.

Темы данной секции:

1.1 Определение требований (Definition of a Software Requirement) – в данном случае подразумевается не процесс определения (извлечения, сбора, формирования, формулирования) требований, а дается само понятие “требования”, как такового, и отмечаются его основные характеристики, например, верифицируемость (проверяемость) требования.

По мнению авторов, необходимо обратить внимание на следующие определения понятия “требование” (на основе работ Вигерса и стандарта IEEE Standard Glossary of Software Engineering Terminology, 1990):

- Условие или возможность, требуемая пользователем для решения задач или достижения целей.
- Условие или возможность, которые должны удовлетворяться системой/компонентом системы или которыми система/компонент системы должна обладать для обеспечения условий контракта, стандартов, спецификаций или др. регулирующими документами.
- Документальная репрезентация (зафиксированное определение, описание) условий или возможностей, перечисленных в предыдущих пунктах.

1.2 Требования к продукту и процессу (Product and Process Requirements) – проводится разграничение соответствующих требований как свойств продукта, который необходимо получить, и процесса, с помощью которого продукт будет создаваться; отмечается, что ряд требований может быть заложен неявно и программные требования могут порождать требования к процессу, например: работа в режиме 24x7 (как бизнес-требование) наверняка приведет к ограничению выбора тех или иных программных средств, платформ развертывания и архитектурных решений; в свою очередь, выбор платформы J2EE (Java 2 Enterprise Edition) и ее реализации в виде конкретного сервера приложений практически наверняка потребует применения модульного тестирования (Unit testing) как практики процесса разработки и JUnit, как инструмента реализации этой практики.

1.3 Функциональные и нефункциональные требования (Functional and Non-functional Requirements) – функциональные требования задают “что” система должна делать; нефункциональные – с соблюдением “каких условий” (например, скорость отклика при выполнении заданной операции); часто функциональные требования представляют в виде сценариев (вариантов использования) Use Case.

По мнению авторов, в определенной степени, систематизируя работы Вигерса, Лефингвелла и Видрига, Коберна, а также других экспертов, необходимо привести классификацию различных категорий (видов) требований и связанных с ними понятий, важнейших с точки зрения их понимания и дальнейшего практического применения:

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

- *Потребности (needs)* – отражают проблемы бизнеса, персоналии или процесса, которые должны быть соотнесены с использованием или приобретением системы.

- *Группа функциональных требований*

    - *Бизнес-требования (Business Requirements)* – определяют высокоуровневые цели организации или клиента (потребителя) – заказчика разрабатываемого программного обеспечения.

    - *Пользовательские требования (User Requirements)* – описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями при помощи создаваемой программной системы.

    - *Функциональные требования (Functional Requirements)* – определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований.

- *Группа нефункциональных требований (Non-Functional Requirements)*

    - *Бизнес-правила (Business Rules)* – включают или связаны с корпоративными регламентами, политиками, стандартами, законодательными актами, внутрикорпоративными инициативами (например, стремление достичь зрелости процессов по CMMI 4-го уровня), учетными практиками, алгоритмами вычислений и т.д. На самом деле, достаточно часто можно видеть недостаточное внимание такого рода требованиям со стороны сотрудников ИТ-департаментов и, в частности, технических специалистов, вовлеченных в проект. Business Rules Group дает понимание *бизнес-правила,* как “положения, которые определяют или ограничивают некоторые аспекты бизнеса. Они подразумевают организацию структуры бизнеса, контролируют или влияют на поведение бизнеса”. Бизнес-правила часто определяют распределение ответственности в системе, отвечая на вопрос “кто будет осуществлять конкретный вариант, сценарий использования” или диктуют появление некоторых функциональных требований.

      В контексте дисциплины управления проектами (уже вне проекта разработки программного обеспечения, но выполнения бизнес-проектов и бизнес-процессов) такие правила обладают высокой значимостью и, именно они, часто определяют ограничения бизнес-проектов, для автоматизации которых создается соответствующее программное обеспечение.

    - *Внешние интерфейсы (External Interfaces)* – часто подменяются “пользовательским интерфейсом”. На самом деле вопросы организации пользовательского интерфейса безусловно важны в данной категории требований, однако, конкретизация аспектов взаимодействия с другими системами, операционной средой (например, запись в журнал событий операционной системы), возможностями мониторинга при эксплуатации – все это не столько функциональные требования (к которым ошибочно приписывают иногда такие характеристики), сколько вопросы интерфейсов, так как функциональные требования связаны непосредственно с *функциональностью* системы, направленной на решение *бизнес-потребностей*.

    - *Атрибуты качества (Quality Attributes)* – описывают дополнительные характеристики продукта в различных “измерениях”, важных для пользователей и/или разработчиков. Атрибуты касаются вопросов портируемости, интероперабельности (прозрачности взаимодействия с другими системами), целостности, устойчивости и т.п.

    - *Ограничения (Constraints)* – формулировки условий, модифицирующих требования или наборы требований, сужая выбор возможных решений по их реализации. В частности, к ним могут относиться параметры производительности, влияющие на

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

выбор платформы реализации и/или развертывания (протоколы, серверы приложений, баз данных, ...), которые, в свою очередь, могут относиться, например, к внешним интерфейсам.

- *Системные требования (System Requirements)* – иногда классифицируются как составная часть группы функциональных требований (не путайте с как таковыми "функциональными требованиями"). Описывают высокоуровневые требования к программному обеспечению, содержащему несколько или много взаимосвязанных подсистем и приложений. При этом, система может быть как целиком программной, так и состоять из программной и аппаратной частей. В общем случае, частью системы может быть персонал, выполняющий определенные функции *системы*, например, авторизация выполнения определенных операций с использованием программно-аппаратных подсистем.

Необходимо сделать несколько важных замечаний по *бизнес-правилам*. Бизнес правила, как таковые, являются предметом пристального изучения различных специалистов в области как бизнес-моделирования, так и программной инженерии в целом. Практика разработки программных требований включает идентификацию и описание бизнес-правил как самостоятельных артефактов. Например, методология RUP выделяет отдельный артефакт Business Rule в рамках дисциплины Business Modeling. Все бизнес-правила, в рамках данной дисциплины, идентифицируются и описываются в документе Business Rules Document. При разработке требований, в сценариях Use Cases обычно включается ссылка на уже описанное бизнес-правило. Скотт Амблер (см. www.agilemodeling.com/artifacts/) так же выделяет бизнес-правило как один из артефактов, который используют в семействе Agile методологий.

В настоящее время разработаны методы и подходы формального представления бизнес-правил, вплоть до формальных языков описания (использование OCL – Object Constraint Language, BRML – Business Rules Markup Language).

Бизнес-правила могут быть не только использованы при определении требований к разрабатываемому ПО, но и могут отдельно оформляться в дизайне ПО (класс или группа классов), отражаясь в конечном итоге в программном коде на определенном языке программирования. Существуют специализированные инструментальные средства и библиотеки, позволяющие разрабатывать и поддерживать приложения, интенсивно использующие бизнес-правила.

Рассматривая бизнес-правила, как артефакты относящиеся к области программных требований можно отметить, вернее дать одно из пояснений, почему БП относят к нефункциональным требованиям: Например, при написании определенного шага в сценарии use case, используется ссылка на бизнес правило: «… система производит расчет стоимости в соответствии с бизнес-правилом BP41 …». В свою очередь данное бизнес-правило может определять алгоритм расчета стоимости. Т.е. налицо «с соблюдением каких условий система делает расчет».

Одним из наиболее известных специалистов по BR является Рональд Росс, автор книги «Principles of the Business Rule Approach» (Ronald G. Ross. «Principles of the Business Rule Approach», 2003).

Наравне с представленной классификацией требований, могут использоваться и другие подходы. Даже в рамках этой классификации, существуют и различные взгляды на ее интерпретацию и детализацию. Например, как результат определения целевой аудитории и в рамках маркетинговой стратегии продвижения тиражируемого решения, возможно определять *высокоуровневые возможности (ключевые характеристики, особенности) – "фичи" (features)* разрабатываемого продукта. Иногда, такие возможности детализируются в смысле функциональных требований в некоторых agile-техниках, например, FDD – Feature-Driven Development (как вы видите вплоть до самого названия целого комплекса практик, метода разработки).

Вигерс, описывает feature как "*множество логически связанных функциональных требований, которые предоставляют определенные возможности для пользователя и удовлетворяют бизнес-целям <организации>*". С точки зрения маркетинга программного обеспечения, как отмечает Вигерс, feature это «группа требований, узнаваемая заинтересованными лицами, которые вовлечены в процесс принятия решения по приобретению ПО – это список <отличительных особенностей или возможностей, характеристик>, присутствующий в описании продукта». В то же время, Леффингвелл и Видриг (D.Leffingwell, D. Widrig, Managing Software Requirements: A Use Case

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

Approach, Second Edition, 2003) определяют feature как "*сервисы, которые оказывает система для удовлетворения одной или более потребностей заинтересованных лиц (stakeholders needs)*". Ими же отмечается, что в реальных проектах могут быть не определены stakeholders needs (а их часто выделяют, особенно если у проекта/продукта есть много заинтересованных лиц со своими потребностями, и эти потребности могут носить взаимоисключающий характер), но существовать features и наоборот, и конечно же возможно существование и stakeholder needs и features – которые при этом должны иметь взаимную трассировку.

Развивая тему features -- Kurt Bittner & Ian Spence в своей книге "Use Case Modeling" (Kurt Bittner, Ian Spence. Use Case Modeling, 2002), дают схожее, хотя и более формальное определение features. Они отмечают, что features могут быть как относящимся к функциональным, так и к нефункциональным требованиям. И могут изменяться от версии к версии продукта.
Анализируя различные источники на предмет работы с features, следует отметить следующее:
С точки зрения инженерии требований, features являются самостоятельным артефактом, который может быть соотнесен как с функциональными требованиями, так и с нефункциональными (в т.ч. с *ограничениями проектирования* или *атрибутами качества*).

Необходимо также отметить, что Features обладают определенным дуализмом в своей интерпретации, зависимым от контекста конкретного продукта – с одной стороны это может быть «тот самый список характеристик, указанный на коробке продукта» в случае создания «коробочного ПО», с другой стороны это может список высокоуровневых возможностей системы, например при заказной разработке ПО автоматизации бизнес-процессов конкретной организации.

Features могут быть разного уровня детализации – от выражения высокоуровневых возможностей системы (например, «Система должна иметь возможность расчета заработной платы …»), до достаточно конкретных требований (например, «Автоматическое уведомление Клиента по e-mail о резервировании товара на складе»)

1.4 Независимые свойства (Emergent Properties) – требования, которые не могут быть адресованы тому или иному компоненты программной системы, но которые должны быть соблюдены, например, в контексте сетевой инфраструктуры или регламентов работы пользователей.

1.5 Требования с количественной оценкой (Quantifiable Requirements) – требования, поддающиеся количественному определению/измерению, например, система должна обеспечить пропускную способность "столько-то запросов в секунду"; в то же самое время, крайне важно понимать, что постановка вопроса (то есть формулирование требования) в форме "система должна обеспечить рост пропускной способности" без указания конкретных количественных характеристик является просто некорректно определенным требованием.

По мнению авторов, при этом, например, требование "система должна вести журнал подключений пользователей" может и должно детализироваться с точки зрения описания информации, которую необходимо сохранять в журнале, однако, такое требование уже не будет являться количественным требованием. А требование с формулировкой "система должна обладать интуитивно-понятным пользовательским интерфейсом" - *непроверяем*. В определенных случаях, по мнению автора книги, это может выглядеть просто издевкой, даже не являясь изначально таковой – все зависит от точки зрения: например, в устах "целевого" пользователя специализированного программного обеспечения – системного администратора, привыкшего работать в kshell (популярной командной оболочке Unix/Linux), объясняющего свои потребности аналитику, фиксирующему запросы пользователя и привыкшего оперировать Microsoft Office ;) Может ли такое требование быть переформулировано и/или детализировано для адекватности интерпретации? Да. Например, так – средний показатель ошибок оператора не должен превышать 2% от объема вводимой информации и 85% пользователей должны дать положительную оценку прототипу пользовательского интерфейса на этапе опытной эксплуатации.

Такие требования должны однозначно отвечать на вопросы, предполагающие ответы с численными величинами – как часто? насколько быстро? в каком количестве? и т.п.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

Большинство требований с количественной оценкой относится к а*трибутам качества*.
В качестве примера можно привести реальное требование, присутствующее в реальном проекте по электронному документообороту: “Система должна производить поиск документов <определенного вида> за время, не превышающее 5 секунд”. Это типичное требование с количественной оценкой, в котором определена верхняя граница диапазона времени, за которое должен быть осуществлен поиск документов. Несомненно, этот атрибут качества системы существует в контексте определенного функционального требования о возможности поиска документов по определенным критериям. И этот контекст или связь должна быть определена либо явно, в рамках иерархии требований, либо посредством трассировки, между требованиями разных видов (функционального и атрибута качества). Примечательно, что Вигерс в своей книге выделяет требования по производительности системы в отдельный вид требований, тем не менее входящих в понятие нефункциональных требований или атрибутов качества.

1.6 Системные требования и программные требования (System Requirements and Software Requirements) – данное разделение базируется на определении “системы”, данном INCOSE (International Council on Systems Engineering) “комбинация взаимодействующих элементов <созданная> для достижения определенных целей; может включать аппаратные средства, программное обеспечение, встроенное ПО, другие средства, людей, информацию, техники (подходы), службы и другие поддерживающие элементы”; таким образом, подразумевается, что система является более ёмким понятием, чем программное обеспечения и включает окружение, в котором функционирует ПО, как таковое; отсюда, естественным образом, вытекают требования к системе в целом и программному обеспечению (или программной системе), в частности. Часто в литературе по управлению требованиями встречается описание системных требований как “пользовательских требований” (user requirements), SWEBOK ограничивает применение понятия “пользовательское требование” требованиями к системе конечных пользователей/заказчиков. Системные требования по SWEBOK, в свою очередь, окружают пользовательские требования (или требования других заинтересованных лиц – stakeholders, например, регулирование полномочий) без указания идентифицируемого источника-человека.

2. Процесс работы с требованиями (Requirements Process)

Данная секция вводит процессы, касающиеся вопросов работы с требованиями, и в определенной степени “сшивает” в единое целое оставшиеся пять секций области знаний, посвященной требованиям к программному обеспечению.

Цель данной темы, в соответствии с SWEBOK, дать понимание того, что такое процесс работы с требованиями, как таковой. В русском языке также устойчиво используется его название как “процесс определения требований”. мы его будем использовать взаимозаменяемым образом, подразумевая весь процесс работы с требованиями по SWEBOK.

Что ж, рассмотрим структуру декомпозиции тем процесса работы с требованиями:

2.1 Модель процесса определения требований:

- Не является дискретным; это постоянно действующий процесс на всех этапах жизненного цикла программного обеспечения. Процесс работы с требованиями инициируется в начале проекта и продолжается на протяжении всего жизненного цикла, в плоть до завершения проекта. Например, функциональные тесты создаются в соответствии с функциональными требованиями к программной системе и обычно выполняются, в том числе, при проведении приемочных испытаний. Как вы уже заметили, автор использовал все же “работа с требованиями” для акцентирования внимания на том факте, что требования не только “определяются” на начальных этапах работ, но и модифицируются и используются во всем жизненном цикле.

- Идентифицирует программные требования как элементы конфигурации (в терминах конфигурационного обеспечения) и контролирует их с использованием тех же практик конфигурационного управления, что и для других активов программных проектов (например,

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

файлов или запросов на изменения).

- Требует адаптации к проектному и/или организационному контексту, в рамках которого ведется соответствующий программный проект.

В частности, тема процесса определения требований касается тех вопросов, которые охватываются в рамках сбора, анализа, специфицирования и утверждения требований ч точки зрения организации этих видов деятельности для различных типов проектов и значимости тех или иных ограничений по отношению к процессу. В большинстве случаев, процесс определения, работы с требованиями выделен в самостоятельный набор и описан как последовательность (сценарии) действий, связанных с ними ролей и непосредственных результатов (их часто называют “артефактами”, например, в RUP – Rational Unified Process), в рамках конкретных методологий разработки программного обеспечения, наиболее популярные из которых мы рассмотрим позднее.

2.2 Участники процессов (Process Actors)

В этой теме вводится понятие “роли” и дается понимание “ролей” для людей, которые участвуют в процессе работы с требованиями (чувствуете отличие между “определением” требований и “работой” с ними?). Таких людей также называют “заинтересованными лицами” (в данном контексте - software stakeholders). Заинтересованное лицо – некто, имеющий возможность (в том числе, материальную) повлиять на реализацию проекта/продукта.

Типичные примеры ролей:

- *Пользователи (Users):* группа, охватывающая тех людей, кто будет непосредственно использовать программное обеспечение; пользователи могут описать задачи, которые они решают (планируют решать) с использованием программной системы, а также ожидания по отношению к атрибутам качества, отображаемые в пользовательских требованиях.

- *Заказчики (Customers):* те, кто отвечают за заказ программного обеспечения или, в общем случае, являются целевой аудиторией на рынке программного обеспечения (образуют целевой рынок ПО);

  Стандарт 12207 (его обзор будет приведен в другой главе) определяет более суженное понятие “заказчика” (обратите внимание – acquirer, а не customer, хотя часто оба термина переводятся на русский язык одинаково) как организацию, которая приобретает или получает систему, программный продукт или программную услугу от поставщика. Здесь возможно использовать такое общее определение: *заказчик* – лицо или организация, получающие прямую или косвенную выгоду от использования продукта. *Клиентами* считают тех заинтересованных лиц, кто требует, оплачивает, выбирает, использует или получает результаты работы программного обеспечения. В этом плане, “заказчик” в понимании стандарта 12207 скорее ближе к “клиенту” в такой интерпретации.

- *Аналитики (Market analysts):* продукты массового рынка программного обеспечения (как и других массовых рынков, например, бытовой техники) не обладают “заказчиками” в понимании персонификации тех, кто “заказывает разработку. В то же самое время, лица, отвечающие за маркетинг, нуждаются в идентификации потребностей и обращению к тем, кто может играть роль <квалифицированных> “представителей” потребителей;

- *Регуляторы (Regulators):* многие области применения (“домены”) являются регулируемыми, например, телекоммуникации или банковский сектор. Программное обеспечение для ряда целевых рынков (в первую очередь, корпоративного сектора) требует соответствия руководящим документам и прохождения процедур, определяемых уполномоченными органами.

- *Инженеры по программному обеспечению, иженеры-программисты (Software Enginner):* лица, обладающие обоснованным интересом к разработке программного обеспечения, например, повторному использованию тех или иных компонент, библиотек, средств и инструментов. Именно инженеры ответственны за техническую оценку путей решения поставленных задач и последующую реализацию требований заказчиков.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

SWEBOK особо подчеркивает, что если невозможно в точности (в оригинале – “perfectly”) удовлетворить требования каждого заинтересованного лица, именно работа инженера включает проведение переговоров и поиск компромисса, приемлемого для ключевых заинтересованных лиц (“стейкхолдеров”) и удовлетворяющего бюджетным, техническим, временным и другим ограничениям проекта. Необходимо понимать, что такая деятельность практически наверняка приведет к изменениям в требованиях, как минимум, на уровне соответствующих приоритетов требований и, следовательно, работ по их реализации.

2.3 Управление и поддержка процессов (Process Support and Management)

Эта тема затрагивает вопросы распределения ресурсов, необходимых для осуществления проектной деятельности, устанавливая контекст для первой секции “Инициация и определение содержания” (Initiation and Scope Definition) области знаний “Управление в программной инженерии” (Software Engineering Management). Основная цель данной темы – обеспечение связи между процессами и деятельностью, определенными в 2.1 “модели процесса определения требований” и вопросами использования проектных ресурсов – стоимостью, человеческими ресурсами, инструментами и т.п.

2.3 Качество и улучшение процессов (Process Quality and Improvement)

Эта тема связана с оценкой качества процессов работы с программными требованиями и улучшением этих процессов. Особое значение данной темы заключается в подчеркивании значимости работы с требованиями, ключевой роли этих процессов для определения стоимостных и временных ресурсов, необходимых для реализации программного проекта, в целом.

Удовлетворение потребностей заказчика является целью любого программного проекта. Соответственно, обеспечение адекватности реализации требований в проекте просто невозможно представить без адекватных процессов работы с ними – начиная со сбора требований, заканчивая проверкой соответствия получаемого программного продукта этим требованиям на всех этапах его создания.

По мнению авторов, улучшение процессов и в частности процессов разработки и управления требованиями должно предваряться формулировкой проблемы. Т.е. нет смысла заниматься улучшением ради улучшения, нужно четко понимать какая в настоящее время есть проблема в работе с требованиями, и насколько эта проблема значима, и только потом приступать к ее устранению, в частности через улучшение процессов. Реальная отечественная практика многих организаций, занимающихся разработкой ПО, показывает, что очень немногие имеют действительно четкое представление о том, каким образом организация работы с требованиям может повлиять на успех компании в целом. Обычно, отечественные компании, в лучшем случае просто документируют требования, выпуская документы, например, Техническое задание по ГОСТ. Но действительно ли в этом документе можно увидеть *требования* – увы. Следуя *только* рекомендациям, которые есть в ГОСТ можно только соответствующим образом оформить разделы, что практически никак не влияет на качество и информативность документа. Вопросы совершенствования процессов – process improvement будут рассматриваться как в главах, посвященных CMMI, так и в других частях этой книги.

Данная тема тесно связана с областями знаний “Качество программного обеспечения” (Software Quality) и “Процесс программной инженерии” (Software Engineering Process). В этом контексте, фокусы обсуждаемой темы – определение атрибутов и метрик качества, а также определение соответствующих процессов в применении к программным требованиям, которые можно свести в три группы практик:

- Покрытие процессов работы с требованиями с точки зрения стандартов и моделей улучшения процессов, в целом;
- Измерение и количественная оценка (benchmarking) процессов работы с требованиями;
- Планирование и реализация процесса улучшения, как такового.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

3. Извлечение требований (Requirements Elicitation)

Данная секция освещает вопросы сбора требований как с точки зрения организации процесса, так и определения источников, откуда поступают требования. Это первая стадия построения видения автоматизируемой проблемной области. Идентификация заинтересованных лиц, их взаимодействия, выполняемых ими бизнес-процессов – все это является ключевыми вопросами, без четкого и однозначного ответа на которые даже не стоит думать об успешности проекта (кстати, не только программного...).

Один из ключевых принципов программной инженерии заключается в обеспечении взаимодействия между пользователями и инженерами. Прежде, чем начинается разработка программного обеспечения, именно специалисты “по требованиям” – аналитики перекидывают тот самый “мостик” между заказчиками и исполнителями, который задает тот уровень коммуникаций и взаимопонимания между ними, который необходим для решения задач проекта.

3.1 Источники требований (Requirement Sources)

Необходимо идентифицировать все возможные источники требований, значимые для решения задач проекта. Только после этого можно определить их влияние на проект. Данная тема касается вопросов понимания информированности источников требований и их значимости.

Данная тема фокусируется на:

- Целях
- Знании предметной области
- Заинтересованных лицах
- Операционном окружении
- Организационной среде

Выделение приоритетов, однозначность требований, передаваемых инженерам, связь между требованиями и их взаимное влияние друг на друга – все это является следствием четкого и однозначного понимания источников требований.

3.2 Техники извлечения требований (Elicitation Techniques)

Идентифицировав источники требований мы не должны “покоится на лаврах”. Даже обладая пониманием того, кто владеет необходимой информацией, мы далеко не застрахованы от проблем, связанных с получением требований, необходимых для дальнейшей работы. Осуществление своей профессиональной деятельности пользователями далеко не гарантирует, к сожалению, способность ясно, четко и однозначно сформулировать то, что они делают и что именно им необходимо для решения их задач сегодня и завтра. Во многом, поэтому, сбор требований, зачастую, превращается в столь тяжелый и часто порождающий конфликты процесс действительно извлечения, “вытаскивания” информации, без которой невозможно переходить к дальнейшим проектным работам. Недопонимание между аналитиком и пользователем, упущение тех или иных аспектов, на первый взгляд кажущихся второстепенными, неоднозначность или тем более некорректность интерпретации информации, полученных от пользователей – все это наиболее типичные причины “сверх-затрат” (времени, денег и т.п.), а иногда, и полного провала проектов.

Существует множество практик и подходов, позволяющих добиться действительно стройной системы требований, отвечающих реальным потребностям и приоритетам заказчиков. Среди них можно выделить следующие:

- *Интервьюрирование* – традиционный подход извлечения требований; не стоит забывать, что получение информации от пользователя “не равно” получению требований; информация должна быть проанализирована и трансформирована в требования, таким образом, информация от пользователя является “входом” в процессы сбора требований, а сами требования – “выходом” этих процессов;

- *Сценарии* – контекст для сбора пользовательских требований, определяющий ответы на вопросы “что если” и “как это делается” в отношении бизнес-процессов, реализуемых

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

пользователями;

- *Прототипы* – отличный инструмент для уточнения и/или детализации требований; существуют разные подходы к прототипированию – от “бумажных” моделей до пилотных подсистем, реализуемых как самостоятельные (в терминах управления ресурсами) проекты или бета-версии продуктов; часто прототипы постепенно трансформируются в результаты проекта и используются для проверки и утверждения требований;

- *“Разъясняющие встречи”* - в оригинале звучит как “facilitated meetings”; достаточно емкий по смыслу термин, пришедший из общей практики менеджмента и базирующийся на идеях сотрудничества заинтересованных лиц для совместного анализа путей решения проблем, определения и предупреждения рисков и т.п. В отличие от “обычного”, с позволения сказать, “мозгового штурма”, как исключительной формы обсуждения тех или иных задач (часто в критические моменты работ над проектом), “запланированный мозговой штурм” – особая форма встреч участников проекта и заинтересованных лиц со стороны заказчика, посвященная обсуждению тех вопросов, ответы на которые не могут быть определены в результате обычных интервью и которые требуют вовлечения большего количества лиц, чем просто пары “пользователь-аналитик”; я позволили себе сконструировать на русском языке этот термин еще и как “запланированный мозговой штурм”, так как такого рода встречи действительно обычно планируются с заданной периодичностью для обеспечения однозначности интерпретации информации, значимой для проекта и, что очень важно – проведения таких встреч до того, как связанные с данными вопросами риски не превратились в реальные проблемы, требующие решения “вчера”, а, следовательно, и дополнительных (изначально незапланированных) ресурсов времени, денег и т.д.;

- *Наблюдение (observation)* – подразумевает непосредственное присутствие аналитиков и инженеров рядом с пользователем в процессе выполнения последним его работ по обеспечению функционирования бизнес-процессов: в определенной степени можно провести аналогию с практикой присутствия представителя заказчика в проектной группе исполнителя (типовая практика в eXtreme Programming “on-site customer” – “присутствующий заказчик”); данная техника является достаточно затратной, но, в то же время, очень эффективной, а иногда – просто незаменимой, особенно, если речь идет о достаточно сложных и взаимосвязанных бизнес-процессах;

Существуют и другие, достаточно эффективные практики, описание которых можно найти в литературе и которые вы, наверняка, сами используете в своей работе (например, Requirements Workshop, Role Playing, Story Boards и т.п.). Некоторые из них будут также упоминаться в контексте конкретных методологий.

4. Анализ требований (Requirements Analysis)

Эта секция посвящена процессам анализа требований, то есть трансформации информации, полученной от пользователей (и других заинтересованных лиц) в четко и однозначно определенные требования, передаваемые инженерам для реализации в программном коде.

Анализ требований включает:
- Обнаружение и разрешение конфликтов между требованиями;
- Определение границ задачи, решаемой создаваемым программным обеспечением; в общем случае - определение “scope” (или “bounds”), границ и содержания программного проекта;
- Детализация системных требований для установления программных требований;

Практически всегда, хотя это явно и не отмечено в описании анализа требований как секции SWEBOK, на практике выделяется и детализация бизнес-требований для установления программных требований. Например, пресловутый режим работы 24x7, сформулированный в виде бизнес-требования, накладывает достаточно жесткие рамки на выбор технологической платформы и архитектурных решений как технических требований к разрабатываемой программной системе..

SWEBOK отмечает, что традиционный взгляд на анализ требований часто сфокусирован или уменьшен до вопросов концептуального моделирования с использованием соответствующих аналитических методов, одним из которых является SADT – Structured Analysis and Design Technique

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

(методология структурного анализа и техники проектирования), знакомый многим по нотациям IDEF0 (функциональное моделирование – стандарт IEEE 1320.1), IDEF1X (информационное моделирование – стандарт IEEE 1320.2, известный также как IDEFObject), часто применяемым как для моделирования бизнес-процессов, так и структур данных, в частности – реляционных баз данных.

Так или иначе, вне зависимости от выразительных средств, которые являются лишь инструментом анализа и фиксирования результатов, результатом анализа требований должны быть однозначно интерпретируемые требований, реализация которых проверяема, а стоимость и ресурсы – предсказуемы.

4.1 Классификация требований (Requirements Classification)

Требования могут классифицироваться по целому ряду параметров, например:

- Функциональные и нефункциональные требования
- Внутренние (с другими требованиями) или внешние зависимости
- Требования к процессу или продукту
- Приоритет требований
- Содержание требований в отношении конкретных подсистем создаваемого программного обеспечения
- Изменяемость/стабильность требований

Другие варианты классификации могут, и часто базируются, на принятых в организации подходах, применяемых методологиях, методах и практиках, а, зачастую, и специфике проектов и даже требованиях заказчиков к процессу разработки и, в частности, определения требований и форме представления результатов их анализа.

4.2 Концептуальное моделирование (Conceptual Modeling)

Разработка модели проблемы реального мира – ключевой элемент анализа требований. Цель моделирования – понимание проблемы, задачи и методов их решения до того, как начнется решение проблемы.

Часто приходится слышать, что прагматичность подхода в отношении программных проектов заключается в “пропуске” этапа (или стадии, фазы) моделирования. В свою очередь, часто ставят знак равенства между моделированием и “этими красивыми квадратиками со стрелочками”. Ни то, ни другое утверждение неверны. Например, в XP и в других гибких (Agile) практиках существуют и истории пользователей, и карточки задач, и процедуры анализа (в частности, связанных с ними “мозговых штурмов”, как запланированных, так и, к сожалению, не очень) , в результате которого мы сформулировали задачи, высокоуровневые возможности - “фичи” продукта (feature - особенность), а также *необходимые модели* (см. [Амблер, 2002]). Объем моделей, их детализация и средства представления могут быть различны. Их выбор базируется и/или диктуется конкретным культурным контекстом организаций, вовлеченных в проект, и практик, применяемых проектной группой. Именно не форма, но сама идея моделирования как попытка упростить и однозначно интерпретировать на концептуальном уровне проблематику деятельности в реальном мире – обязательная составляющая как управления требованиями, так и программной инженерии, в целом.

Среди факторов, которые влияют на выбор модели, метода и детализации ее представления, степени связанности с программным кодом и другими вопросами:

- Природа проблемы (проблемной области)
- Экспертиза и опыт инженеров
- Требования заказчика к процессу
- Доступность методов и инструментов
- Внутрикорпоративные стандарты и регламенты
- Культура разработки

В любом случае, моделирование рассматривается в программном контексте, а не только с точки зрения бизнес-задач как таковых, Это обусловлено необходимостью понимания операционного и

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

системного контекста, то есть окружения, в котором программная система будет реально использоваться и которое накладывает свои, иногда достаточно жесткие ограничения.

Вопросы моделирования тесно связаны с применяемыми методами и подходами. Однако, частные методы или нотации, как отмечается в SWEBOK, так или иначе следуют распространенным в индустрии практикам и тяготеют к тем формам, с которыми связаны накопленный опыт и подтвержденные общепринятой практикой знания. SWEBOK отмечает, что могут быть разработаны различные виды моделей, включающие потоки работ и данных, модели состояний, трассировки событий, взаимодействия пользователей, объектные модели, модели структур данных, и т.п. Кстати, именно такая ситуация сложилась с UML, все чаще воспринимаемым в качестве общепринятого или de-facto стандарта в моделировании и включающем целый комплекс моделей (в UML 2.0 включено 14 моделей, представленные в двух группах – статические модели и поведенческие), связанных и объединенных общей архитектурой, на основе концепции метамоделей.

По мнению автора, современное состояние стандарта UML (унифицированного языка моделирования Unified Modeling Language, разрабатываемого консорциумом OMG – www.omg.org) версии 2.0 вполне позволяет говорить о расширении его применимости в “чистом” бизнес-моделировании. На фоне богатства выразительных средств, появления соответствующего инструментального обеспечения работы с UML 2.0, длительной истории успешного применения стандарта UML 1.x, инструментов на его основе и повсеместного использования UML в области объектно-ориентированного анализа и проектирования не только аналитиками, но архитекторами и разработчиками ПО, можно с уверенностью говорить о смещении фокуса индустрии программного обеспечения в сторону UML и отходу (как минимум, частичному) от IDEF, в применении к аналитической деятельности. Темпы такой “миграции”, конечно, зависят от степени консервативности взглядов конкретных специалистов-аналитиков. Однако, давление рынка, требование унификации, в частности, выразительных средств описания активов проектов в рамках всей проектной команды – те причины, по которым, по мнению автора, аналитики, не воспринявшие UML-ориентированный тренд, могут оказаться за бортом серьезных корпоративных ИТ-проектов. Даже на фоне “неприятия” UML некоторыми игроками рынка, критическая масса знаний и практик по его применению уже оказалась достаточно велика, чтобы игнорировать его применение. В то же самое время, не стоит воспринимать UML как панацею – это касается любой технологии, практики или подхода. Создан, активно развивается и уже поддержан индустрией стандарт BPMN – Business Process Management Notation (см. www.bpmi.org). Таким образом, все определяется конкретным “культурным” контекстом. Просто надо помнить об этом и оставаться “прагматиками”, в положительном понимании этого слова, не теряя креативности в повседневной деятельности.

Необходимо отметить, что на практике наблюдается тенденция разделения вопросов определения требований и моделирования. Это, например, заметно в современных методологиях, таких как RUP (Rational Unified Process), где работа с требованиями и моделирование/проектирование – суть две разные дисциплины (об этом мы будем говорить в соответствующей главе).

4.3 Архитектурное проектирование и распределение требований (Architectural Design and Requirements Allocation)

Считается, что создание архитектуры программных решений является обязательным элементов успешности таких проектов. Архитектурное проектирование перекрывается с программным и системным дизайном (проектированием) и иллюстрирует насколько сложно провести четкую грань между различными аспектами проектирования. Данная тема работы с программными требованиями тесно связана с секцией “Структура и архитектура программного обеспечения” (Software Structure and Architecture) области знаний “Проектирование программного обеспечения” (Software Design). Во многих случаях, инженеры действуют как архитекторы, потому как процессы анализа и выработки требований зависят от программных компонент, создаваемых для решения поставленных заданными требованиями задач, призванных, в конечном счете, добиться реализации поставленных перед проектом целей.

Архитектурное проектирование очень близко к концептуальному моделированию. Непосредственное отображение бизнес-сущностей реального мира на программные компоненты не является обязательным. Во многом поэтому и существует такое разделение между моделированием и проектированием. В принципе, можно говорить о том, что деятельность по моделированию в большей степени касается того, ”что” надо сделать, а архитектура - “как” это будет реализовано.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

Существует ряд стандартов и общепринятых практик, связанных с архитектурным проектированием. Среди них наиболее популярны:

- Стандарт IEEE 1471-2000 “Recommended Practice for Architectural Description of Software Intensive Systems”
- Модель Захмана – Zachman Framework (www.zifa.com)
- TOGAF – The Open Group Architecture Framework (www.opengroup.org/architecture/)

Важно заметить, что ни в коем случае не надо путать архитектурные рекомендации (Architectural Guidelines) с практиками и стандартами архитектурного проектирования. Одни (например, Federal Enterprise Architecture Framework FEAF - !!!) дают рекомендации по конкретным архитектурно-технологическим решениям. Другие концентрируются именно на том, чему стоит уделить внимание при создании архитектуры, как ее описать и детализировать, и что из себя представляет *архитектура*, как таковая (например, ISO 15704 Industrial Automation Systems – Requirements for Enterprise-Reference Architectures and Methodologies).

5. Спецификация требований (Requirements Specification)

На инженерном жаргоне, да и в терминологии ряда методологий, устоялся термин “software requirements specification” (SRS) – спецификация программных требований. Для сложных систем, на самом деле, существует целый комплекс спецификаций, документов, которые являются результатом сбора и анализа требований, их моделирования и архитектурного проектирования. Эти документы систематически анализируются, в них вносятся изменения, они пересматриваются и утверждаются. Чаще всего, для описания комплексных проектов (в части требований) используется три основных документа (спецификации):

- Определение системы (system definition)
- Системные требования (system requirements)
- Программные требования (software requirements)

5.1 Определение системы (System Definition Document)

Данный документ, часто называемый как “спецификация пользовательских требований” (user requirements specification) или “концепция” (concept <of operations>), описывает системные требования. Содержание документа определяет высокоуровневые требования, часто – стратегические цели, для достижения которых создается программная система. Принципиальным моментом является то, что такой документ описывает требования к системе с точки зрения области применения - “домена”. Соответственно, изложение требований в нем ведется в терминах прикладной области. Документ описывает системные требования вместе с необходимой информацией о бизнес-процессах, операционном окружении с точки зрения бизнес-процедур и организационных и других регламентов. Примером стандарта для создания и структурирования такого документа является IEEE 1362 “Concept of Operations Document”.

5.2 Спецификация системных требований (System Requirements Specification)

В сложных проектах принято разделять спецификацию системных требований (system requirements) и спецификацию программных требований (software requirements). При таком подходе программные требования порождаются системными требованиями и детализируют требования к компонентам и подсистемам программного обеспечения. Документ описывает программную систему в контексте системной инженерии (system engineering), идеи которой кратко описаны в Главе 12 SWEBOK “Связанные дисциплины программной инженерии”. Строго говоря, спецификация системных требований описывает деятельность системных инженеров и выходит за рамки обсуждения SWEBOK. Стандарт IEEE 1233 является одним из признанных руководств по разработке системных требований. И, как уже отмечалось ранее, не стоит забывать о том, что понятие *система*, в общем случае, охватывает программное обеспечение, аппаратную часть и людей. Системная инженерия (см. материалы INCOSE – www.incose.org), в свою очередь, самостоятельная и не менее объемная дисциплина чем программная инженерия. SWEBOK рассматривает системную инженерию как важную связанную дисциплину. Ну а системные требования – один из элементов реального связывания различной инженерной деятельности - программной и системной.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

5.3 Спецификация программных требований (Software Requirements Specification - SRS)

Часто эту спецификацию называют “требованиями к программному обеспечению”. Все же, учитывая существование дисциплин системной и программной инженерии, мы используем термин “программные требования”, как более точно подходящий по смыслу по моему мнению.

Программные требования устанавливают основные соглашения между пользователями (заказчиками) и разработчиками (исполнителями) в отношении того, что будет делать система и чего от нее не стоит ожидать. Документ может включать процедуры проверки получаемого программного обеспечения на соответствие предъявляемым ему требованиям (вплоть до содержания планов тестирования), характеристики, определяющие качество и методы его оценки, вопросы безопасности и многое другое. Часто программные требования описываются на обычном языке. В то же время, существуют полуформальные и формальные методы и подходы, используемые для спецификации программных требований. В любом случае, задача состоит в том, чтобы программные требования были ясны, связи между ними прозрачны, а содержание данной спецификации не допускало разночтений и интерпретаций, способных привести к созданию программного продукта, не отвечающего потребностям заинтересованных лиц. Стандарт IEEE 830 является классическим примером стандарта на содержание структурирование и методы описания программных требований – “Recommended Practice for Software Requirements Specifications”.

По мнению авторов, в документацию на требования не следует вносить элементы дизайна системы (скажем, логическую модель базы данных). А вот сценарии использования Use Case часто включают в спецификацию требований наравне с трассировкой (traces) к соответствующим моделям в форме диаграмм, например, к UML Use Case, UML Activity, BPMN и т.п. . Говоря о написании спецификаций требований, то есть одно серьезное заблуждение, которое делают обычно неопытные аналитики – это фактическая подмена требований как таковых, моделями графического пользовательского интерфейса, т.е. когда в документы-спецификации требований просто включаются «картинки» пользовательского интерфейса с небольшими пояснениям. Это отнюдь не означает, что с заинтересованными лицами и в частности с пользователями, не следует вообще обсуждать дизайн GUI, часто это имеет смысл делать, но для этого существует, например, прототипирование. Мне довелось изучить многие документы требований в разных организациях и практически все они имели одни и те же проблемы:

1. *Терминологическая неопределенность*. Часто используются термины, которые обладают многозначностью, и такие термины не определены в глоссариях, чтобы можно было четко понять, что конкретно автор имеет в виду в данном случае (это не всегда бывает понятно из контекста). Как пример можно привести собственно использование (и, что немаловажно, понимания!) термина «требования». Под этим ёмким термином можно понимать как требования к бизнес процессам, так и функциональные или нефункциональные требования к ПО вообще. Интересно, что на уровне многих стандартов (к сожалению, в основном, англоязычных) прописано использование тех или иных глаголов, форм и структурирование предложений, описывающих требования – например использование глаголов (will, shall, should, may, can – перечислены в порядке “убывания директивности”). Действительно, “программный модуль X отсылает уведомление на e-mail адрес пользователя...” несет, мягко говоря, иную смысловую нагрузку, чем “отсылается сообщение”.

2. *Отсутствие представления о классификации требований*. *Подмена одних категорий требований – другими и смешение требований* (например, такое часто случается с функциональными требованиями, бизнес-требованиями и бизнес-правилами). Как результат – создаваемые документы тяжело читать и извлекать полезную для разработки ПО информацию. Зачастую в одном абзаце, можно встретить перемешанные как описания необходимой функциональности и тут же элементы предполагаемого пользовательского интерфейса, который должен воплотить разработчик. Или проектные решения, например, использование таблиц баз данных или полей. И помимо этого, содержится несистематизированная и фрагментарная информация о бизнес-процессах организации. Все это скрывает истинные требования к разрабатываемому ПО, что в свою очередь затрудняет как разработку, так и согласование требований. Корректная и однозначная интерпретация требований и анализ влияний становятся практически неосуществимыми, что напрямую сказывается на адекватности удовлетворения потребностей заказчика/пользователей.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

3. *Фокусировка на деталях пользовательского интерфейса.* В документах встречается акцентирование не на необходимой функциональности, а на деталях пользовательского интерфейса.

4. *Излишнее акцентирование внимания на деталях реализации.* Попытка отразить в документе с требованиями к создаваемому ПО не ЧТО должна делать система, а то КАК она это будет делать. Это одна из ключевых проблем. Во многом, поэтому, часто выделяют внутренние технические требования к системе, которые не проходят аттестации со стороны пользователей и разрабатываются не аналитиками, а архитектором и ведущими разработчиками уже на этапе проектирования – software design (см. следующую область знаний SWEBOK).

5. *Слабая формализация бизнес-процессов.* В документах перемешивается описание бизнеса и требования к ПО, что приводит сложностям в понимании сути и общему пониманию как должна быть спроектирована система.

6. Проверка требований (Requirements Validation)

Определение требований напрямую связано с процедурами проверки (verification) и утверждения (аттестации - validation, как это сформулировано в ГОСТ Р и ISO/IEC 12207). Вообще говоря, принято считать, что требования описаны неполностью, если для них не заданы правила V&V (*verification&validation – проверка и аттестация*), то есть не определены способы проверки и утверждения. Процедуры проверки являются отправной точкой для инженеров-тестировщиков и специалистов по качеству, непосредственно отвечающих за соответствие получаемого программного продукта предъявляемым к нему требованиям.

К сожалению, как уже комментировалось выше, часто, в крупных организациях вместо полноценной проверки сути и содержания документов, все сводиться к так называемому "нормоконтролю" -- когда проверка документов требований заключается в проверке на соблюдение принятых стандартов внешнего оформления документа (отступы и размеры поля, подписи таблиц/рисунков и т.п.), но никак ни оценки качества требований. И совершенно неверно считать такой "нормоконтроль" полноценной проверкой требований.

Если стандарты жизненного цикла описывают как *правильно создавать* продукт, то стандарты (в том числе, внутрикорпоративные) отвечают за создание *правильного продукта*, то есть того продукта, который соответствует ожиданиям пользователей и других заинтересованных лиц.
Для достижения этой цели применяется ряд практик, в том числе, представленных ниже.

6.1 Обзор требований (Requirements Review)

Один из распространенных методов проверки требований - инспекция или обзор требований. Суть его заключается в том, что ряд лиц, вовлеченных в проект (для крупных проектов – специально выделенные специалисты), “вычитывают” требования в поисках необоснованных предположений, описаний требований, допускающих множественные интерпретации, противоречий, несогласованности, недостаточной степени детализации, отклонений от принятых стандартов и т.п.

Вопросы обзора требований, вообще говоря, имеют непосредственное отношение к теме качества, поэтому они также описываются в области знаний SWEBOK “Качество программного обеспечения” (Software Quality) в теме 2.3 “Обзор и аудит” (Review and Audit).

6.2 Прототипирование (Prototyping)

В общем случае, прототипирование подразумевает проверку инженерной интерпретации программных требований и извлечение новых требований, неопределенных или неясных на ранних итерациях сбора требований. Существует множество подходов к прототипированию, как с точки зрения детализации, так и того, чему уделяется внимание при прототипировании. Наиболее часто прототипы создаются для оценки способа реализации пользовательского интерфейса и проверки архитектурных подходов и решений.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

При всей безусловной полезности прототипирования для обеспечения проверки требований и решений, необходимо понимать, что с прототипированием связан ряд вопросов способных привести к негативным последствиям или, как минимум, работам, требующим дополнительного времени и средств. Среди *возможных* негативных последствий прототипирования стоит выделить следующие:

- Смещение внимания с целевых функций прототипа и, как следствие, неудовлетворенность пользователей огрехами прототипа, отсутствием стоящей за ним реальной функциональности (для прототипов пользовательского интерфейса), ошибками в прототипе и т.п.
- Превращение прототипа в реальную систему за счет постоянного добавления новых свойств и функциональности “для проверки” – часто бывает нарушена архитектурная целостность, необеспечена необходимая масштабируемость и качество получаемого программного продукта;

Здесь, авторы хотели бы добавить и еще одну типичную проблему - переключение внимания заинтересованных лиц на эргономику и детали дизайна графического пользовательского интерфейса, при начальной цели построения прототипа для выявления функциональных и иных требований и наоборот. Проблема не во внимании пользовательскому интерфейсу, проблема в подмене, если так можно выразиться, функциональной составляющей пользовательским интерфейсом (вспомните, как часто вы сами говорили или слышали – “я не о ‘кнопочках’ и ‘окошках’, я о задаче ...”).

Конечно, ясно, что эти факторы можно превратить и в положительные стороны прототипа. Кроме того, не стоит считать что прототип это всегда нечто, воплощенное в код. Прототипом пользовательского интерфейса может быть, например, просто “прорисованный” на бумаге или в электронной форме набор переходов между экранами/диалоговыми окнами системы (кстати, это подход, часто используемый в Agile-практиках, но отнюдь *не заменяющий* требований к системе).

Так или иначе, выбор того или иного метода прототипирования, да и самого факта такого способа проверки требований или технологических идей, должен основываться на временных и других имеющихся ресурсах, опыте в прототипировании и, конечно, степени сложности создаваемой программной системы.

6.3 Утверждение модели (Model Validation)

Утверждение или аттестация модели связана с вопросами обеспечения приемлемого качества продукта. Уверенность в соответствии модели заданным требованиям может быть закреплена формально со стороны пользователей/заказчика. В то же время, проверка и аттестация модели, например, объектно-ориентированного представления бизнес-сущностей и связей между ними может быть проверена с той или иной степенью использования формальных методов, например, статического анализа (поиск связей и путей взаимодействия между описанными объектами и выделение различного рода несоответствий). Это – другая сторона утверждения модели.

6.4 Приемочные тесты (Acceptance Tests)

Требования должны быть верифицируемы. Требования, которые не могут быть проверены и аттестованы (утверждены) – это всего лишь “пожелания”. Именно так они буду восприниматься разработчиками, даже в случае их высокой значимости для пользователей. Если описанное требование не сопровождается процедурами проверки – в большинстве случаев говорят о недостаточной детализации или неполном описании требования и, соответственно, спецификация требований должна быть отправлена на доработку и если необходимо, должны быть предприняты дополнительные усилия, направленные на сбор требований.

Можно говорить о том, что процедура анализа требований считается выполненной только тогда, когда все требования, включенные в спецификацию, обладают методами оценки соответствия им создаваемого программного продукта. Чаще всего столь строгое ограничение на степень законченности спецификации накладывается на функциональные требования и атрибуты качества (например, время отклика системы).

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

Идентификация и разработка приемочных тестов для нефункциональных требований часто оказывается наиболее трудоемкой задачей. Для ее решения обычно “ищут точку опоры”, то есть возможность взгляда на описываемые требования с количественной точки зрения, в плоть до переформулирования и большей степени детализации описания таких требований.

Дополнительная информация, связанная с приемочными тестами представлена в области знаний SWEBOK “Тестирование программного обеспечения” (Software Testing) в описании 2.2.4 “Тесты соответствия” (Conformance testing).

7. Практические соображения (Practical Considerations)

Первый уровень декомпозиции секций данной области знаний напоминает описание последовательности действий. Это, безусловно, упрощенный взгляд на процесс работы с требованиями. Данный процесс, точнее, комплекс процессов, охватывает весь жизненный цикл программного обеспечения. Управление изменениями и сопровождение, поддержка актуальности требований и их реализации – ключ к успешным процессам программной инженерии.

Далеко не каждая организация обладает культурой документирования и управления требованиями. Особенно часто это встречается в молодых небольших компаниях, выводящих на рынок новые продукты и обладающие “сильным вижином”, четким пониманием целей, для которых создается продукт, но не имеющих достаточно ресурсов и, во многом поэтому считающих, что динамизм – залог успеха. Постепенно такие компании вырастают, проекты – усложняются и, как следствие складывается ситуация, когда отследить все необходимые требования в неформальной форме уже просто невозможно. Эта тема практически неисчерпаема. Многие средние по масштабам компании пытаются сохранить тот же вровень гибкости и динамизма, который применялся во времена рождения компании, когда она еще была “стартапом” (start-up – название молодых компаний, которые раскручивали свои проекты во времена интернет-бума конца 90-х и которое прижилось для вновь образующихся малых бизнесов, растущих не столько на внешних инвестициях, сколько на идеях и упорстве ее создателей). Так или иначе, динамизм присущ не только компаниям, но и продуктам, самим требованиям к ним. Управление изменениями, концепцией, видением продукта не может быть хаотическим – история индустрии однозначно это показывает. Поэтому отношение к управлению требованиями как к постоянно действующему бизнес-процессу – абсолютно обоснованный подход, требующий применения определенных практик. В противном случае, мы практически гарантировано столкнемся с темни негативными последствиями, которые не раз описывались и упоминались выше.

7.1 Итеративная природа процесса работы с требованиями (Iterative Nature of the Requirements Process)

В большинстве случаев, понимание и интерпретация требований продолжает эволюционировать в процессе проектирования и разработки программного обеспечения. Кроме того, требования часто меняются в силу изменений бизнес-контекста для которого создается и в котором эксплуатируется программное обеспечение. Необходимо понимать неизбежность изменений и планировать шаги по уменьшению проблем, связанных с изменениями. В то же самое время, современные практики гибкой разработки говорят о том, что необходимо концентрироваться только на том, что требует внимания “прямо сейчас”, не закладываясь на предупреждение всех возможных рисков, в том числе связанных с изменениями, включая изменения требований. Говорить о том, какой подход – предупреждение или реагирование является гарантировано приводит к успеху – сложно сказать. Более того, если кто-то однозначно настаивает только на одной из идей и полностью отвергает другую – это профанация. Восприятие изменений и возможность их своевременной обработки - вопрос способности проектной команды работать в условиях постоянно меняющихся условий, принимаемых архитектурных решений и многих других культурных, технологических и организационных факторов. Так или иначе, понимание меняющейся природы требований – один из факторов адекватного реагирования на сами изменения, а, следовательно, и возможности успешного завершения проекта.

7.2 Управление изменениями (Change Management)

Управление изменениями – одна из ключевых тем управления требованиями. Необходимость определения процедур для обработки изменений совсем не то же самое, что и их детальная формализация. Такие процедуры необходимы. Им посвящена тема управления изменениями в приложении к требованиям. В то же время, рассматривать изменение требований в отрыве от других процессов, по меньшей мере, кажется странным. Соответственно, данный вопрос является составной частью управления изменениями и конфигурациями программного обеспечения (Software Configuration and Change Management, SCCM), которое сегодня принято называть просто конфигурационным управлением (Software Configuration Management, SCM), подразумевая, что это не только вопросы контроля версий, но управление всеми активами проекта, включая код, требования, запросы на изменения – change requests (в том числе, отчеты об ошибках – defect или bug reports), задачи (в терминах проектного менеджмента) и т.п.

Общий комплекс вопросов конфигурационного управления рассматривается в области знаний SWEBOK “Управление конфигурациями программного обеспечения” (Software Configuration Management).

7.3 Атрибуты требований (Requirements Attributes)

Требования должны состоять не только из описания того, что необходимо сделать, но и содержать информацию, необходимую для интерпретации требований и управления ими. Например, с функциональными требованиями часто ассоциируют сценарии Use Case (как в текстовом, так и графическом представлении) и, в то же время, функциональные требования часто трансформируются в задачи в терминах проектного управления, с которыми связаны параметры законченности (например, в процентах), ответственности (например, кто является “владельцем” требования, кто из инженеров назначается исполнителем или принимает на себя обязанности, связанные с реализацией заданной функциональности, как это принято, например, в XP или FDD). Примеров существует множество и, в зависимости от применяемых практик и методов, сложившейся проектной и организационной культуры, спектр атрибутов может меняться достаточно широко, практически, неограниченно.

Необходимо также помнить, что к обсуждаемым атрибутам также относятся параметры, связанные с классификацией требований (см. выше тему 4.1 “Классификация требований”). В свою очередь, принадлежность к тому или иному классу (категории, типу, виду) требований означает не только семантику того, “чему посвящены” требования (функциональности, параметрам качества и т.п.), но и комплекс атрибутов, общий для всех требований данного класса.

По мнению авторов, можно провести параллель между требованиями и записями (строками) в реляционной базе данных, где каждая запись обладает набором атрибутов (столбцов). В определенном смысле, можно и необходимо говорить о том или ином уровне *атомарности* требований (что не исключает связей между требованиями), представляемой такой метафорой.

7.4 Трассировка требований (Requirements Tracing)

Трассировка требований обеспечивает связь между требованиями и отслеживание источников требований. Трассировка является фундаментальной основой проведения анализа влияния (impact analysis) при изменении требований, помогая предсказывать эффект от внесения таких изменений. Трассировка предполагает направленную связь (представляется в виде сложного направленного ациклического графа) между требованиями, то есть зависимости.

Требования (B) обладают обратной зависимостью (то есть вторичны) по отношению к требованиям (A) и заинтересованным лицам, которые являются источником либо образуют причину появления рассматриваемых требований (B). И, наоборот, требования (A) трассируются напрямую к тем требованиям (B) и элементам дизайна (например, модели или, в общем случае, кода, запросов на изменения и т.п.), которые порождаются или удовлетворяют требованиям (A).

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

7.5 Измерение требований (Measuring Requirements)

С практической точки зрения, обычно полезно иметь нечто, позволяющее определить “объем” требований для заданного (создаваемого) программного продукта. Это число полезно для исследования “масштабов” изменений в требованиях, оценки стоимостных характеристик (cost estimation) разработки и поддержки программной системы, опосредовано – оценки продуктивности разработки и эффективности поддержки на этапах реализации требований и внесения изменений и т.п.

Измерение объема функциональности (Functional Size Measurement, FSM) техника такого рода численной оценки, определена на концептуальном уровне в стандарте IEEE 14143.1. Стандарты ISO/IEC и другие источники описывают частные методы FSM (например, модель COCOMO II для оценки стоимости, например, может использоваться в тесной связи с методами функциональных точек – functional points для оценки масштабов функциональности, то есть требований, предъявляемых заданной программной системе).

Дополнительная информация по стандартам и подходам в оценке масштабов представлена в области знаний “Процесс программной инженерии” (Software Engineering Process).

В дополнение к практическим соображениям, представленным в SWEBOK, на фоне общей тенденции разработки моделей <оценки> зрелости, стоит отметить, что существуют определенные работы и по созданию различных моделей зрелости требований. Например, наиболее популярная модель зрелости в индустрии программного обеспечения – CMMI включает разный объем и содержание работ по определению и управлению требованиями для уровней зрелости 2 и 3.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com