Математические структуры и моделирование
2011, вып. 23, с. 76–85

УДК 004.414.38

МЕТОДИКИ ПРИОРИТИЗАЦИИ ТРЕБОВАНИЙ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

А.В. Непомнящих

Рассматриваются методики приоритизации требований в проектах по разработке ПО и различные аспекты внедрения данных методик.

Введение

По статистике два из трёх проектов терпят неудачу [4]. Таким образом, практически любой проект можно считать «безнадёжным» в той или иной степени.

По исследованиям 2006 года [5] одним из ключевых факторов успеха является правильная работа с требованиями.

Известный эксперт в области менеджмента программных проектов Эдвард Йордон [1] считает, что самое важное для успешного завершения проекта – это приоритизация требований:

> ...Если в этот момент вы решили, что у вас нет времени читать всю книгу, скажу только одно слово, которое может окупить время, потраченное на чтение предисловия: приоритетность (triage). Если вы участвуете в безнадёжном проекте, почти наверняка окажется недостаточно ресурсов, чтобы реализовать всю функциональность и возможность ПО, которые требуются конечному пользователю, в рамках утверждённого плана и бюджета. Так или иначе придётся решать, какие возможности следует реализовывать в первую очередь, а какими можно пожертвовать. Действительно, некоторые из незначительных возможностей не будут реализованы никогда, поэтому самое лучшее – это дать им спокойно умереть собственной смертью. Другие возможности являются достаточно важными, но также относительно легко реализуемыми, например, с помощью поставляемой библиотеки классов или используемых вами CASE-средств. Говоря языком медиков, эти возможности выживут сами по себе. Успех или неудача безнадёжного проекта зачастую зависит от способности проектной команды выделить критические функции и возможности,

Омский государственный университет им. Ф.М. Достоевского
E-mail: anepomnyaschih@gmail.com

> которые нельзя реализовать, не вкладывая значительные ресурсы и энергию...
>
> *Предисловие*

> ...Один из ключевых моментов в безнадёжных проектах состоит в том, что некоторые требования не просто останутся невыполненными до официального срока завершения проекта, но и не будут выполнены никогда. Если предположить, что известное правило «80-20» справедливо, то проектная команда сможет добиться 80 процентов «эффекта» от разработанной системы, реализовав 20 процентов требований к ней – при условии правильного выбора этих 20 процентов. И, поскольку пользователю, как правило, не терпится получить работающую систему задолго до того срока, который проектная команда считает разумным, то он может взять эти готовые 20 процентов, приступить к их использованию и навсегда позабыть об оставшихся 80 процентах функций системы...
>
> *Гл. 5.1 Концепция «triage»*

Другой эксперт, Джоэл Спольски также описывает этот факт [2]:

> Если бы мы не «отложили» все те «супер-важные» требования, разработка Excel 5 заняла бы вдвое больше времени и включала бы 50% бесполезной функциональности, которую бы требовалось сопровождать, для обратной совместимости, до конца времён.

В данной работе рассматриваются возможные проблемы при введении приоритизации в практику выполнения проектов по разработке ПО и предлагаются методы решения данных проблем.

1. Что предлагают эксперты?

Йордон:

> Все требования должны быть распределены по трём группам: «необходимо выполнить», «следует выполнить», «можно выполнить».

И проектная группа в первую очередь должна сконцентрировать усилия на требованиях первой группы, затем второй и затем, если останутся ресурсы, на третьей группе.

Те же принципы используются и в современных «гибких» методологиях разработки ПО:

Кент Бек [6]:

> ...XP подразумевает, что возможности с наивысшим приоритетом будут реализованы в первую очередь... В рамках XP заказчик должен определить наименьший допустимый набор возможностей, которыми должна обладать минимальная работоспособная версия программы, имеющая смысл с точки зрения решения бизнес-задач...

В методологии SCRUM есть понятие *очереди требований*. Таким образом, заказчик лишается возможности поставить каким-либо требованиям равный приоритет и убирается неявный конфликт, когда команду заставляют реализовывать все требования сразу.

В производственной системе Тойота также есть подобное понятие – *вытягивание* и *система канбан* [3]. Данные инструменты служат для той же цели: процесс-потребитель «вытягивает» работу у предыдущего процесса в цепи. Заказчик – у тестировщика, тестировщик – у разработчика, разработчик – у аналитика, аналитик вытягивает требования у заказчика. У процесса-потребителя есть только ограниченное количество карточек, по которым он может заказать продукт. Так как заказчик тоже является потребителем в данной цепи, то ему приходится выбирать, какие требования должны быть реализованы в первую очередь.

2. Определения

Требования бывают разных уровней. Требования верхнего уровня — это, фактически, бизнес-цели заказчика системы. По мере проработки аналитиками требования детализируются и уточняются. Проработанные требования нижнего уровня содержат описание отдельных частей системы и взаимосвязей между ними. Самыми сильными являются связи между требованиями верхнего уровня и их потомками.

Таким образом, имеется *дерево требований*.

Запросом на изменение (ЗИ) будем называть некий запрос в *системе управления проектом (СУП)*, в котором содержится описание функциональности, которую необходимо реализовать. В данном запросе также должно быть указано – исходя из какого набора требований нужна данная функциональность. То есть имеется прямая взаимосвязь между запросами на изменение и требованиями к системе.

Соответственно, бывают следующие типы ЗИ:

1. Новое требование — когда необходимо реализовать ещё одну ветвь дерева требований.

2. Улучшение — когда необходимо внести изменения в систему в соответствии с изменением некоторых требований.

3. Дефект — когда обнаружено несоответствие работы системы набору требований.

3. Препятствия внедрению приоритизации требований со стороны заказчика

Если рассмотреть природу появления требований, то этот процесс грубо можно описать так:

1. У заказчика появляется необходимость в решении некоторой реальной проблемы. Или идея, подразумевающая некоторую конкретную выгоду.

2. Пока заказчик ищет исполнителя, он привыкает к своей идее, и со временем она начинает обрастать всё новыми и новыми уточнениями и улучшениями. «Раз уж я в это ввязался, то почему бы не включить в проект ещё и вот это, и это?» — думает заказчик.

3. Идея обрастает совсем экстремальными и инновационными требованиями, которые до сих пор никем не были реализованы.

Если принять известное правило «80-20», то из вышесказанного можно сделать два вывода:

1. 80% выгоды находятся в 20% начальных требований (1).

2. 80% рисков содержатся в 20% поздних требований (3). Но эти требования, по сути, не решают изначальной задачи, а, в основном, «украшают» её.

Когда команда начинает просить заказчика выставить приоритеты, возникает следующая проблема: заказчику трудно отказываться от любых своих требований, особенно третьей группы (3), которые часто являются нестандартными, творческими, его собственными идеями — плодами его фантазии.

А как мы уже поняли, от них-то и следует отказаться в первую очередь, если возникнет проблема со сроками или ресурсами.

Во-вторых, заказчик может, в принципе, не понимать, как можно сортировать требования, ведь он представляет продукт цельным, и, например, уже согласована спецификация конечного продукта.

В такой ситуации нужно начинать переговоры и постараться объяснить заказчику необходимость сортировки требований:

1. Нужно объяснить, что требования поддаются сортировке (см. 3.1.).

2. Также хорошо помогает концепция очерёдности требований: «если бы вы могли получать требования последовательно, по очереди, в каком порядке вы бы хотели их получить? Какие требования помогли бы вам получить максимальную выгоду уже сейчас?»

3. Вместо слов «давайте уберём это требование» лучше использовать слова «давайте выполним его немного позже».

4. Команда и заказчик должны осознавать, что с любым проектом по разработке ПО всегда сопряжены огромные риски и доля неопределённости: по статистике [4] две третьих проектов по разработке ПО проваливаются. Таким образом, «если случится некоторая проблема, то мы можем что-то не успеть сделать к сроку. Чем бы вы могли пожертвовать в такой безвыходной ситуации? Да, мы полностью понесём ответственность по контракту, но по факту срок может быть сорван, и мы хотим, чтобы вы наименее от этого пострадали».

И даже если заказчик согласился разбить требования на три вышеупомянутых группы, он всё равно будет интуитивно пытаться всем требованиям поставить приоритет «необходимо». Даже если при старте проекта требования были упорядочены, то после нескольких первых итераций большинство требований снова получают высший приоритет – заказчику приходится каждую итерацию делать нелёгкий выбор, и он склоняется вообще его не делать.

Поэтому лучше всего зарекомендовал себя следующий подход: предоставляем заказчику список требований с их примерными оценками и просим упорядочить их по важности и желаемому порядку поступления. Не выставить числовые приоритеты, а именно упорядочить. И предупреждаем, что в этом порядке мы и будем над ними работать. То есть если возникнут непредвиденные трудности, то к сроку будет поставлено скорее то, что находится наверху списка. Таким образом, мы неявно заставляем заказчика упорядочить их линейно.

Когда заказчик начинает понимать, что он может регулировать ход проекта и менять порядок выполнения требований, он может начать использовать это неправильно и выставлять в начало требования третьей группы (см. раздел 3.), которые являются интересными для него, но не дают на самом деле большой выгоды.

В таком случае помогает:

1. Увеличение длительности итераций – у заказчика уже меньше средств для «игр» с требованиями. И если он хочет получить в результате итерации продукт, который можно продавать, то ему придётся включить некоторые требования первой группы (см. раздел 3.).

2. Постоянное отслеживание объёма оставшихся работ и оставшихся ресурсов и сроков. Эти показания передаются заказчику, и делаются рекомендации — можно ли ещё менять требования [7].

3.1. Независимость требований

Как мы уже поняли, приоритизация очень важна. Но как можно дать заказчику возможность менять очерёдность реализации требований, если все требования взаимосвязаны? Ведь порядок реализации требований будет сильно влиять на оценки сроков и стоимости реализации. А без оценок заказчик не может сортировать требования, т.к. сортировка выполняется как раз по соотношению выгода/стоимость для каждого требования.

1. Чтобы оценить соотношение выгода/стоимость, заказчику достаточно знать грубые оценки. Высокая точность оценок здесь не так важна, как при определении состава итерации, т.е. сроков поставки.

2. Если при старте проекта требования были упорядочены и выделены наиболее важные, то в дальнейшем данный порядок будет меняться незначительно. Более того, скорее всего часто будет меняться порядок только в верху списка, когда при старте очередной итерации заказчику нужно

будет выбрать, что в неё включить. Либо когда реализацию некоторого требования решили отложить.

3. Обычно требования в пределах одного слоя одной ветви дерева требований являются независимыми и порядок их реализации не важен. Важно только, когда данный слой в принципе начнёт разрабатываться (см. раздел 4.2.).

4. После выполнения каждого требования будет производиться переработка кода, которая позволит сохранить достаточно гибкую архитектуру проекта. Во-вторых, команда видит все будущие требования, даёт им оценки и, следовательно, представляет, куда движется проект. Таким образом, меньше вероятности столкнуться с ситуацией, когда очередное требование нельзя реализовать в рамках текущей архитектуры без её значительной переработки.

4. Препятствия внедрению приоритизации требований со стороны команды

4.1. Иерархическая структура работ

Заметим, что на практике дерево требований часто путают с иерархической структурой работ.

> *Иерархическая структура работ (ИСР)* – это ориентированная на результаты иерархическая декомпозиция работ, которые должна выполнить команда проекта для достижения целей проекта и создания требуемых результатов; на каждом более низком уровне ИСР представляет всё более детальное описание работ по проекту.
>
> *PMBOK, [8]*

Дело в том, что оценка стоимости проекта, согласно [8], выполняется именно по ИСР. Поэтому часто заказчик в описании стоимости проекта получает именно ИСР.

Здесь появляется ещё одна проблема: заказчик не может приоритизировать ИСР! Он может приоритизировать только требования! Поэтому оценки стоимости должны выставляться для требований. Как это сделать, описано в разделе 3.1.

4.2. Проблема первой итерации

Когда начинается разработка очередной ветви системы, слоя в этой ветви, то обычно необходимо подготовить некоторую инфраструктуру для данного слоя. Например, требование – это «рисование графических примитивов», а подтребования — «овал», «прямоугольник», «линия». Таким образом, в начале реализации необходимо будет подготовить возможность для рисования примитивов вообще, а затем уже разработать конкретные примитивы.

Такая необходимость возникает всегда, когда речь заходит о подобии. Программист стремится всё универсализировать, всю подобную функциональность выделить в отдельные классы и использовать их повторно.

Потому при старте проекта первая итерация, как правило, критична тем, что нужно создать функциональность, которая будет приносить выгоду бизнесу, но при это большую часть итерации приходится потратить не на создание полезного, а на создание того самого каркаса, инфраструктуры.

Предлагается два решения данной проблемы:

1. Использовать готовые каркасные решения. Фреймворки и CMS.

2. Использовать подход методологии eXtreme Programming (XP): "keep it simple". Не нужно делать того, что потом может не понадобиться: не стоит выделять каркас до того, как он пригодился. В каждой итерации делается переработка кода, на которую выделяется специально отведённое время.

Если вернуться к примеру, можно было бы поискать готовые компоненты рисования векторной графики. В таком случае мы берём на себя риск, что переделка компонента под наши конкретные нужды может занять много времени.

Если готовых компонентов не нашлось, нужно начать реализовывать решение для овала самостоятельно. Предположим, что про прямоугольник и линию заказчик ещё ничего не сказал. Тогда реализация рисования овала уместилась бы в одном классе. Затем нам сообщают про прямоугольник и линию. Вот тогда мы перерабатываем код, выделяем инфраструктуру рисования примитивов в абстрактные базовые классы и добавляем необходимые примитивы.

Казалось бы, быстрее было бы сразу заготовить необходимую инфраструктуру и её использовать. Но мы *не знали заранее*, понадобится ли она нам. А абстракции ухудшают читаемость и сопровождаемость кода. Возможно, мы бы потратили время на то, что потом будет не нужно, а поддерживать эту функциональность всё равно пришлось бы.

В данном же случае всего мы затратили времени больше, но получили большую выгоду — график добавления ценности в систему стал более гладким. Исчезла ситуация когда заказчик долгое время не получает полезной функциональности из-за того, что разработчики делают архитектуру чересчур гибкой, хотя большая часть этой гибкости, скорее всего, в будущем и не понадобится [9].

5. Проблемы менеджмента при внедрении приоритизации требований

В процессе исследования данной проблемы удалось выделить нижеследующие группы технических средств.

5.1. Багтрекеры

К багтрекерам относятся такие инструменты, как JIRA, Redmine, Bugzilla, Assembla.

Данные системы содержат в себе ЗИ различных типов (см. раздел 2.) в общей неупорядоченной базе. Хотя каждый ЗИ имеет различные атрибуты, в том числе и атрибут «приоритет», но очередь из них построить сложно.

Главная проблема данных систем — нет чёткой приоритизации. Команда и заказчик должны искусственно поддерживать актуальность приоритетов задач.

Зато данные системы сильно распространены, хорошо конфигурируются и могут поддерживать процесс самых различных проектных команд, будучи достаточно привычными и для заказчиков.

5.2. Системы поддержки «гибкой» разработки

К ним относятся: Pivotal Tracker, Agile Rally и др.

Содержат средства поддержки стандартных практик «гибкой» разработки. Например, строго ориентированы на итеративность. То есть все требования должны быть строго разнесены по итерациям, и нет возможности не использовать данную практику.

Главным положительным моментом для нас является то, что требования в данных системах представляют собой упорядоченный список – чем выше в списке, тем выше приоритет. Таким образом, заказчику и команде просто приходится сортировать требования и выделять важнейшие.

Отрицательные моменты:

1. Жёстко закреплённый процесс разработки. Низкие возможности конфигурирования. Проектной команде придётся подстраивать свой процесс под используемый инструмент.

2. Данные системы поддерживают достаточно небольшие команды (до 9 человек). Для большей команды просто не хватит возможностей пользовательского интерфейса данных систем. Также в данных системах практически отсутствуют средства документирования требований – можно указать лишь краткое описание задачи, что является проблемой для большой команды, где нет возможности часто встречаться всем её членам.

3. Мало распространены и потому непривычны большинству команд и заказчиков.

5.3. Требование в системе управления проектом

Таким образом, чтобы иметь возможность приоритизации, мы получаем достаточно жёсткое определение нового требования:

1. Каждое требование должно храниться отдельно.

2. У требования должен быть атрибут «приоритет» (см. 5.5.), либо требования должны храниться в линейно упорядоченном списке.

3. Требование должно быть понятным заказчику, чтобы он мог оценить выгоду от его реализации.

4. Требование должно иметь атрибуты – оценки затрат бюджета и времени.

5. Требование должно быть именно требованием, а не элементом ИСР.

Этих критериев достаточно для возможности сортировки требований по соотношению выгода/стоимость.

5.4. Дефекты

Дефект – это найденное несоответствие функциональности и требований. По сути, дефект – это незавершённая часть функциональности. Поэтому, когда найден дефект, должен быть переоткрыт соответствующий ЗИ с комментарием, описывающим дефект.

Но обычно найденные дефекты оформляются в системе отдельно, т.к. поиск дефектов — отдельная от самого программирования процедура, которая выполняется не разработчиками, а отдельными участниками проекта для повышения качества тестирования. Они могут не знать, какое конкретное требование привело к данному дефекту, потому оформляют его отдельно.

С приоритизацией дефектов проблем не возникает, кроме единственной: какой приоритет им ставить? Ведь заказчику хочется получить новую функциональность. Поэтому часто бывает, что дефекты получают достаточно низкие приоритеты, а ПО – соответствующее качество... Некоторые рекомендации по этому поводу есть в [10].

5.5. Глобальные и локальные приоритеты

В самых популярных системах управления проектами на сегодняшний день, таких как Atlassian JIRA, Microsoft Visual Studio Team Suite, т.е. Bugtrackers, отсутствует понятие очерёдности требований.

Данная проблема этих систем может быть решена следующим образом. Создаются две группы приоритетов — глобальные и локальные приоритеты.

Глобальные приоритеты выставляются при первичной сортировке требований. Данных приоритетов может быть сколько угодно – всё зависит от способности заказчика к упорядочению своих пожеланий. Минимально необходимое количество, в соответствии с рекомендацией Э. Йордона, это три приоритета: «необходимо», «важно», «желательно». Каждое требование проекта имеет некоторый глобальный приоритет.

При формировании списка требований на очередную итерацию берутся требования с наивысшим глобальным приоритетом, и им присваивается некоторый локальный приоритет. Минимальное необходимое количество локальных приоритетов: «обещанное» (Promised) и «желательное» (Stretched). Локальный

приоритет необходим как сигнал команде, какие из требований могут быть вычеркнуты в случае срабатывания рисков во время итерации. Сроки поставки сообщаются для всех требований, включённых в итерацию, но обговаривается, что «желательные» могут быть опущены в случае срабатывания рисков (см. 3.).

6. Рискованные требования — первыми

Таким образом, обоснована необходимость приоритизации и её осуществимость. Теперь нужно рассмотреть требования в контексте рисков. Дело в том, что с реализацией каждого требования сопряжены определённые риски, то есть потенциальные проблемы, которые могут осуществиться с определённой долей вероятности.

Если взглянуть на приоритизацию с точки зрения рисков, то проблема такова: если какие-то риски, сопряжённые с определёнными требованиями, реализуются на раннем этапе, то, возможно, будет выгоднее отменить проект, т.к. для реализации оставшихся требований не хватит бюджета или времени.

Литература

1. Йордон Э. Путь камикадзе. М. : Лори, 2008. 289 с.
2. Spolsky J. Evidence Based Schedulling. URL: `http://www.joelonsoftware.com/items/2007/10/26.html` (дата обращения: 5.11.2010).
3. Лайкер Дж., Майер Д. Практика дао Toyota. Руководство по внедрению принципов менеджмента Toyota. М. : Альпина Паблишерз, 2009. 584 с.
4. Standish Group. CHAOS 2009. URL: `http://www.standishgroup.com/newsroom/chaos_2009.php` (дата обращения: 5.11.2010).
5. Standish Group. The Standigh Group Report. URL: `http://www.projectsmart.co.uk/docs/chaos_report.pdf` (дата обращения: 5.11.2010).
6. Бек К. Экстремальное программирование. СПб. : Питер, 2002. 224 с.
7. Дорофеев М. Статистика и Окно неопределенности. URL: `http://www.slideshare.net/Cartmendum/traininglabs09-part-3-of-4` (дата обращения: 5.11.2010).
8. Руководство к своду знаний по управлению проектами. Четвёртое издание // Project Management Institute. 2008. 241 с.
9. Поппендик М., Поппендик Т. Бережливое производство программного обеспечения. М. : Вильямс, 2010. 256 с.
10. Спольски Дж. Тест Джоэла: 12 шагов к лучшему коду. URL: `http://russian.joelonsoftware.com/Articles/TheJoelTest.html` (дата обращения: 5.11.2010).