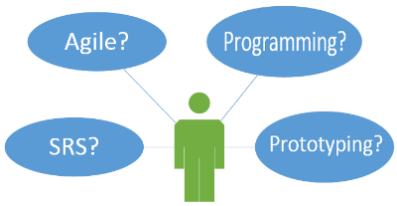

ЧТО ДЕЛАТЬ, когда даже Agile «не рулит»?

Часть I

Нет, ну не хотят заказчики тратить драгоценное время на подробное описание своего бизнеса и бизнес-целей, у них же от этого жизненно важные бизнес-процессы нарушиться могут! А уж про человекообразное описание смысла того, что каждый потенциальный пользователь с системой делать должен – так это, вообще, сверх меры, сколько времени на это угрохать надо! Да и скучно очень, кроме ежедневных текущих бумажек, ещё и всякие внеплановые производить. И главное, кто этим заниматься будет – сам начальник или его подчиненные, все же до единого заняты? Обычно, выделит руководство «контактное лицо» для коммуникаций с разработчиками, но повседневных обязанностей с него не снимет, и будет это лицо любые отговорки находить, чтобы от свалившегося на него «счастья» увильнуть.

Казалось бы, в самый раз бизнес-аналитику свои навыки продемонстрировать, если бы не одно НО: редкий начальник захочет, чтобы кто-то со стороны, по рабочим местам таскался и к людям в рабочее время с глупыми вопросами приставал, т.е. «*интервью*» проводил или целыми днями на рабочих местах сидел и шпионил – кто и чем занят, «*наблюдением*» называется… А, уж опросы фокус-групп или «мозговые штурмы», может они для Америки хороши, а у нас совершенно не серьёзно, просто цирк с фокусами какой-то, а за него еще и денежки платить?

Часто заказчик считает, что за финансы, которые он пообещал, всё само и без его участия должно делаться, а в успехе проекта больше разработчики заинтересованы, так как это, определённо, их заработок, то пусть они сами и крутятся. Поэтому, если расскажут в двух словах цели, упомянут о пользователях и снабдят попавшимися под руку бумагами (прайсы, рекламные проспекты, бланки и т.д.), то считайте, что на этом миссия по постановке требований со стороны заказчика уже выполнена.

И так, есть бизнес-цели, идеи и желания инвесторов, имеются ограниченные финансы, а ясных и конкретных требований нет, добиться их конкретизации от заказчика в ряде ситуаций практически не реально, процесс их выявления, систематизации и управления требованиями может оказаться сложным и длительным, сравнимым с разработкой самого программного продукта. Что делать – отказаться или продолжать проект, писать спецификации и управлять требованиями или нет, какую методологию разработки выбрать? Какие имеются при этом варианты?

Вариант 1. Выяснение требований. При этом процесс разработки требований оформляется отдельным соглашением, в результате которого создаётся и согласуется с заказчиком спецификация. Возможно, это была бы не плохая идея, но с одной стороны мысль, что нужно платить деньги за разработку требований, а не за готовое приложение вряд ли воодушевит заказчика, с другой стороны, будет ли полученная еще до разработки ПО спецификация полезной и ускорит ли она разработку?

На ресурсе [1] приводится 6 причин по которым не стоит писать функциональные спецификации. Приведу их очень кратко в формулировке источника:

1. *Спецификация — это фикция. Она не имеет никакого отношения к реальности.*
2. *Задача спецификации — угодить всем.*
3. *Спецификация — лишь иллюзия соглашения.*
4. *Спецификация заставляет вас принимать самые важные решения тогда, когда вы меньше всего знаете о проекте.*
5. *Использование спецификаций приводит к обрастанию ненужной функциональностью.*
6. *Спецификация не позволяет вам развиваться, меняться и оценивать пройденный путь.*

От себя добавлю, оценить правильность этих формулировок в полной мере может тот, кто хотя бы однажды, прошел полный путь от разработки требований до отладки и тестирования готового приложения.

Вариант 2. Применяем Agile. Например, Scrum – чудо-методология, почти «*серебряная пуля*», её используют при неполных и нестабильных требованиях. Главное убедить заказчика, что равномерная периодическая оплата за реализованный функционал лучше, чем поэтапные финансовые вливания за стандартную отчётность плюс плата за то, что получится по завершении проекта.

Но и тут оказывается всё не так просто. Как утверждается в [2], чудес не бывает, и: «***Bullshit на входе = bullshit на выходе»***, *«…в итоге требования родятся в муках итераций. Что-то будет сделано не так, …и не обойтись без переделок. …Переделок будет много. И тогда полетят сроки, начнётся спешка». Примечание: bullshit – да извинят меня уважаемые читатели, бычье дерьмо пер. с англ.*

Это значит, что представитель заказчика (*Product Owner*) должен подготовить по меньшей мере *sprint backlog* — своеобразную замену спецификации, но не на весь проект, на ближайшие две-три недели разработки, что тоже чревато выше описанными проблемами. Ещё следует учесть, что Scrum подходит далеко не для всех команд, а только для мотивированных, самоорганизующихся и кросс-функциональных, к тому же и объём разработки играет не последнюю роль.

Вариант 3. *«**Пилите гири, Шура!**»* [3] или манифест крутых программеров от Зеда Шоу *(Zed A. Shaw, программист)*: *«**Пиши код, твою мать!**»* [4]. Привожу полностью без купюр, но и без дословного перевода, так как *Motherfucker* – это уже перебор, желающие могут найти перевод, но там, к сожалению, тем более перебор…

Рисунок 1. Манифест от Зеда Шоу

Смысл данного послания, выполненного в противовес знаменитому Agile-манифесту в том, что главное – писать код и это единственная правильная методология, всё остальное не имеет ценности и придумано чтобы создать видимость работы десятка менеджеров, не умеющих писать код.

> *Примечание: очевидно, по мнению автора манифеста в число этих менеджеров входят и аналитики!*

В дополнение к манифесту Зед Шоу предлагал пошаговую инструкцию по выполнению замечательной методологии «Пиши код…»:

1. *Выпишите список всей той хрени, которая должна входить в ваш продукт;*
2. *Реализуйте всю эту хрень путём написания кода;*
3. *Протестируйте написанное;*
4. *Если что-то работает неправильно – то напишите правильный код.*

После п. 4 немедленно переходите к п. 1. Повторять до достижения необходимого результата.

Всё предельно просто и не лишено смысла. Действительно, только тестируя само приложение можно в полной мере оценить его функциональность. Нет ни парного программирования, ни передвижения листочков по доске, ни ежедневных *standup*-митингов. Вопрос только в том, сколько раз потребуется переписать код, чтобы устранить ошибки и удовлетворить заказчика и сколько на это потребуется времени?

Вариант 4. **Прототипирование vs кодирования**. Эта возможность появилась с развитием программ для создания функциональных прототипов (Axure RP, Prototyper и д. р.), на основе которых можно в динамике оценить и протестировать не только функции системы, обеспечивающие её бизнес-логику, но и функционал, относящийся к UX.

Уровни «видимости» функциональных требований

Чтобы представлять какой функционал выявляют и конкретизируют динамические интерактивные прототипы, рассмотрим иерархию функциональных требований, подобную известной от К. Вигерса [5], представив всё в виде гигантского айсберга, вершина которого может быть скрыта в облаках, а основание уходит в глубину (рис. 2).

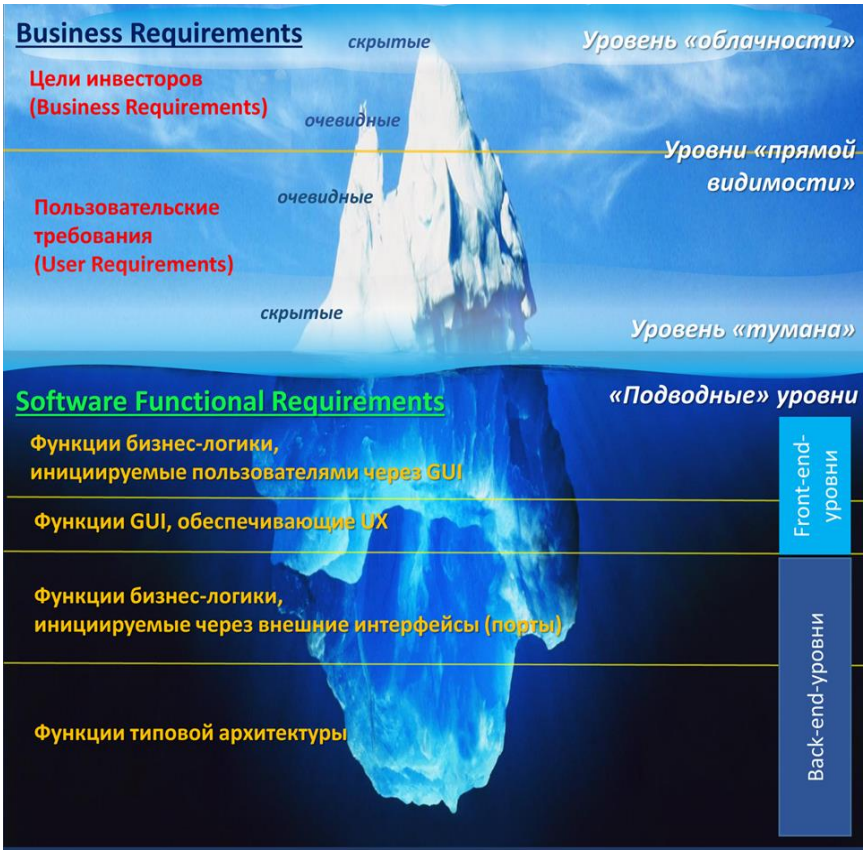

Рисунок 2. Уровни «видимости» функциональных требований

Всё, что на рисунке показано выше «уровня моря», в соответствии с К. Вигерсом, относится к группе бизнес-требований (*Business Requirements*), всё то, что ниже – к группе функциональных требований к ПО (*Software Functional Requirements*). Следует заметить, что требования, входящие в обе эти группы, по своей сути являются функциональными и их можно представить в форме:
*<**субъект**> должен делать <**действие** или **деятельность**>.*

В случае *User Requirements* субъектом будет пользователь, а в случае *Software Functional Requirements,* субъектом будет выступать программная система. Указанная выше форма позволяет отличать функциональные от нефункциональных требований. Последние в данной статье не рассматриваются.

И так, самый высокий уровень – цели инвесторов (*Business Requirements*). Они наиболее абстрактные, могут быть не всегда видимыми, так как не во всех случаях инвестор открывает свои истинные цели. Мне приходилось сталкиваться с ситуацией, когда озвучивались одни, а достигались другие цели, поэтому на рисунке их, частично, могут прикрывать «облака».

Пользовательские требования (*User Requirements*) являются реализацией высокоуровневых бизнес-целей. Они бывают очевидными и понятными, но не редко, скрытыми и непонятными (рис. 2 уровень «тумана»), в том числе для самих заказчиков и пользователей. Это означает, что их надо выявлять и конкретизировать. Пользовательские требования обуславливают объём разработки, однако из-за высокого уровня абстракции и направленности на пользователя, они не являются архитектуро-определяющими, так как от них сложно перейти к архитектуре ПО, которая затем реализуется в программном коде. Для этого и нужны более конкретные и детальные функциональные требования.

Первая группа функциональных требований – это функционал ***front-end***-уровня.

К нему следует отнести функции, отвечающие за исполнение бизнес-логики приложения, инициируемые пользователями (физическими лицами) через графический интерфейс (GUI). Они являются реализацией большей части пользовательских требований, их необходимо выявлять и конкретизировать, трассировать, они могут предполагать различные решения, которые следует согласовать с пользователями, и могут изменяться в процессе разработки. Ими занимаются аналитики требований и проектировщики, они включаются в спецификацию функциональных требований, на их основе разрабатывается уникальная часть архитектуры приложения. Их реализация предстаёт перед пользователями в виде кнопок, полей, списков, меню и других элементов GUI.

К *front-end* относится и дополнительный функционал, реализуемый через элементы GUI, но связанный не с бизнес-логикой, а с UX, т. е. с удобством интерфейса, исключением ошибок пользователя, обеспечением защиты целостности системы, информационной поддержкой пользователей и другой функционал, в соответствии с принципами UX [6]. Этот функционал может не включаться в спецификацию, а определяется специалистом по проектированию UX, он имеет своё отражение в разрабатываемой архитектуре приложения.

Практически весь функционал, относящийся к front-end можно, выявить, детализировать и согласовать с заказчиками с помощью динамических прототипов.

Далее следуют более «глубокие» уровни, относящиеся ***back-end***, т.е. к функциям, реализуемым в большей мере серверной, а не клиентской частью приложения.

К этой группе можно отнести функции, поддерживающие бизнес-логику, но инициируемые не физическими лицами, а внешними системами через порты, либо временем (таймером), также функционал, отвечающий за исполнение стандартных процедур, например, управление базами данных, управление внешними устройствами и т. д.

Функционал, относящийся к back-end, за редким исключением, не поддается прототипированию с помощью браузерных html-прототипов. Но большей частью, это готовые решения, которые реализуются с помощью стандартной архитектуры. Обычно, имеется и их реализация в программном коде. Это «епархия» проектировщика и данный функционал по большей части скрыт от аналитика.

Вместо заключения

Имеется достаточно много методологий разработки ПО, от полного хаоса до крайне формализованных. Каждая из них имеет свои плюсы и минусы, а её конкретный выбор часто зависит от «правила трёх П»: Продукта (который мы создаём), Проекта (в рамках которого должны работать) и Персонала (который имеется у нас в наличии). При этом полезно помнить одно условие: *при разработке ПО, всё без исключения, должно делаться для того, чтобы ускорить процесс создания программного кода, обеспечив приемлемое для заказчика качество продукта*.

Поэтому, лучше творчески подходить к используемым методам и методологиям и, приступая к работе над очередной спецификацией, описанием или диаграммой задать себе вопросы: «Для кого я это делаю?», «Улучшит ли это качество продукта?» и «Ускорит ли это написание кода?».

Свой практический опыт подобного подхода я постараюсь изложить во второй части статьи.

Ссылки на источники:

1. Эссе из книги «Getting Real» (http://gettingreal.37signals.com/), перевод ресурс «Хорошие IT-статьи» http://factorized.tumblr.com/
2. А. Минкевич http://dev.by/blogs/main/kogda-agile-ne-rabotaet-neskolko-primerov
3. И. Ильф, Е. Петров «Золотой телёнок»
4. Зед Шоу (Zed A. Shaw) Manifesto http://programming-motherfucker.com/
5. Вигерс К. Разработка требований к программному обеспечению. Второе издание Microsoft Press, 2004
6. Nielsen J.“Ten Usability Heuristics”, www.nngroup.com/articles/ten-usability-heuristics

Автор:

Николай Киреев, старший преподаватель ИИТ БГУИР, индивидуальный предприниматель (разработка ПО, студия **WebMax.BY**), руководитель проектов, аналитик (freelancer)

Skype: nousy123,
e-Mail: nousy@mail.ru