Лекция 2: Жизненный цикл программного обеспечения, требования

Юрий Литвинов
y.litvinov@spbu.ru

На основе материалов Тимофея Александровича Брыксина, бывшего доцента кафедры системного программирования СПбГУ

1. Жизненный цикл программного обеспечения

В первой лекции говорилось о том, что сложную программную систему построить «простыми» методами невозможно. Ее разработка с неизбежностью будет тоже сложной деятельностью. Привести такое дело к успеху возможно на основе общих принципов работы со сложными системами: организовав его в виде набора компонент, используя разные уровни абстракции. Выделить компоненты можно, определив набор задач, которые нужно решить для достижения конечной цели. Для решения каждой такой задачи организуется вспомогательная деятельность, к которой можно также применить декомпозицию на отдельные, более мелкие деятельности, и т.д.

В качестве примеров деятельностей, которые нужно проводить для построения программной системы, можно привести проектирование — выделение отдельных модулей и определение связей между ними с целью минимизации зависимостей между частями проекта и достижения лучшей его обозримости в целом; кодирование — разработку кода отдельных модулей; разработку пользовательской документации, которая необходима для достаточно сложной системы. Однако для корректного с точки зрения инженерии и экономики рассмотрения вопросов создания сложных систем необходимо, чтобы были затронуты и вопросы эксплуатации системы, внесения в нее изменений, а также самые первые действия в ходе ее создания — анализ потребностей пользователей и выработка решений, «изобретение» функций, удовлетворяющих эти потребности. Без этого невозможно, с одной стороны, учесть реальную эффективность системы в виде отношения полученных результатов ко всем сделанным затратам и, с другой стороны, правильно оценивать в ходе разработки степень соответствия системы реальным нуждам пользователей и заказчиков.

Весь период существования ПО, связанный с подготовкой к его разработке, разработкой, использованием и модификациями, начиная с того момента, когда принимается решение разработать/приобрести/собрать из имеющихся компонентов новую систему, или приходит сама идея о необходимости программы определенного рода, до того момента, когда полностью прекращается всякое ее использование, называют *жизненным циклом ПО*.

В ходе жизненного цикла создаются и перерабатываются различного рода артефакты — создаваемые человеком информационные сущности, документы в достаточно общем

смысле, участвующие в качестве входных данных и получающиеся в качестве результатов различных деятельностей. Примерами артефактов являются: модель предметной области, описание требований, техническое задание, архитектура системы, проектная документация на систему в целом и на ее компоненты, прототипы системы и компонентов, собственно, исходный код, пользовательская документация, документация администратора системы, руководство по развертыванию, база пользовательских запросов, план проекта, и пр.

На различных этапах в создание и эксплуатацию ПО вовлекаются люди, выполняющие различные роли. Каждая роль может быть охарактеризована как абстрактная группа заинтересованных лиц, участвующих в деятельности по созданию и эксплуатации системы и решающих одни и те же задачи или имеющих одни и те же интересы по отношению к ней. Примерами ролей являются: бизнес-аналитик, инженер по требованиям, архитектор, проектировщик пользовательского интерфейса, программист-кодировщик, технический писатель, тестировщик, руководитель проекта по разработке, работник отдела продаж, конечный пользователь, администратор системы, инженер по поддержке и т.п.

Похоже, что общую структуру жизненного цикла любого ПО задать невозможно, поскольку она существенно зависит от целей, для которых это ПО разрабатывается или приобретается, и от решаемых им задач. Тем не менее, часто определяют основные элементы структуры жизненного цикла в виде *модели жизненного цикла ПО*. Модель жизненного цикла ПО выделяет конкретные наборы видов деятельности (обычно разбиваемых на еще более мелкие активности), артефактов, ролей и их взаимосвязи, а также дает рекомендации по организации процесса в целом. Эти рекомендации включают ответы на вопросы о том, какие артефакты являются входными данными у каких видов деятельности, а какие появляются в качестве результатов, какие роли вовлечены в различные деятельности, как различные деятельности связаны друг с другом, каковы критерии качества полученных результатов, как оценить степень соответствия различных артефактов общим задачам проекта и когда можно переходить от одной деятельности к другой и т.д.

Стоит отметить, что идея модели процесса появилась в программной инженерии очень давно, в те времена, когда считалось, что удачная модель — самое главное, что способствует успеху разработки. Позднее пришло осознание, что существует множество других аспектов (принципы управления и разработки, структура команды и т.д.), которые должны быть определены согласованно друг с другом.

1.1. Модели жизненного цикла

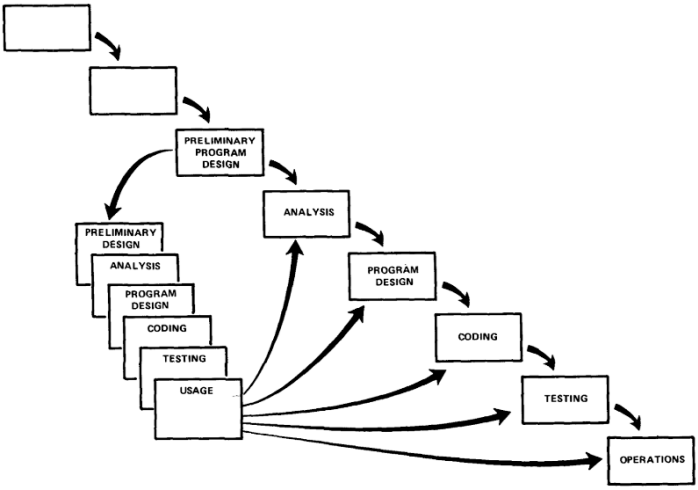
1.1.1. Водопадная модель

Наиболее широко известной и применяемой долгое время оставалась так называемая каскадная или водопадная (waterfall) модель жизненного цикла, которая, как считается, впервые четко сформулирована в 1970 году Винстоном Ройсом в работе «Managing the Development of Large Software Systems»[1] и впоследствии запечатлена в стандартах министерства обороны США в семидесятых-восьмидесятых годах XX века. Эта модель предполагает последовательное выполнение различных видов деятельности, начиная с выработки требований и заканчивая сопровождением, с четким определением границ между этапами, на которых набор документов, созданный на предыдущей стадии, передается в качестве входных данных для следующей. Таким образом, каждый вид деятельности выполняется

[1] http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf (дата обращения: 25.02.2022).

на какой-то одной фазе жизненного цикла. «Классическая» каскадная модель предполагает только движение вперед по этой схеме: все необходимое для проведения очередной деятельности должно быть подготовлено в ходе предшествующих работ. Отмечалось, что возвраты могут катастрофически увеличить стоимость проекта и сроки его выполнения. Например, если при тестировании обнаруживаются ошибки проектирования или анализа, то их исправление часто приводит к полной переделке системы. Этой моделью допускались возвраты только на предыдущий шаг: например, от тестирования к кодированию, от кодирования к проектированию и т.д.

Ещё в рамках этой модели (и об этом редко говорят) было введено прототипирование, то есть предлагалось разрабатывать систему дважды, чтобы уменьшить риски разработки. Первая версия — прототип — позволяет увидеть основные риски и обоснованно принять главные архитектурные решения, оценить технические ограничения. На создание прототипа отводилось до одной трети времени всей разработки. Также Ройс настаивает на непосредственном вовлечении заказчика в процесс разработки. Это требование мы будем встречать и в последующих моделях жизненного цикла.



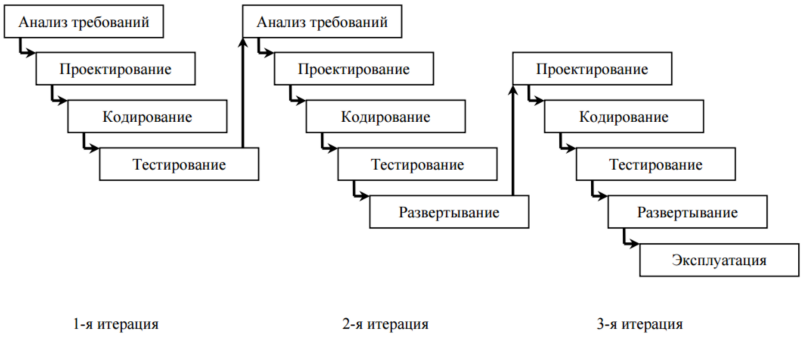
В качестве плюсов данной модели можно отметить относительную простоту планирования проекта (чёткие этапы с чёткими границами). Также на разных этапах нужны разные специалисты, а значит, их можно привлекать лишь на определённое время.

Работать в соответствии с этой моделью можно, только если удается предвидеть заранее возможные особенности хода проекта и тщательно собирать и анализировать информацию на первых этапах, с тем, чтобы впоследствии можно было пользоваться их результатами без оглядки на возможные изменения.

Сейчас данная модель продолжает использоваться на практике — для небольших, простых проектов, в хорошо формализованных проектах (и в тех, и в других есть чётко определённое ТЗ, которое вряд ли будет меняться) или при разработке типовых систем, где итеративность не так востребована. С ее помощью удобно отслеживать разработку и осуществлять поэтапный контроль за проектом.

1.1.2. Инкрементально-итеративные модели

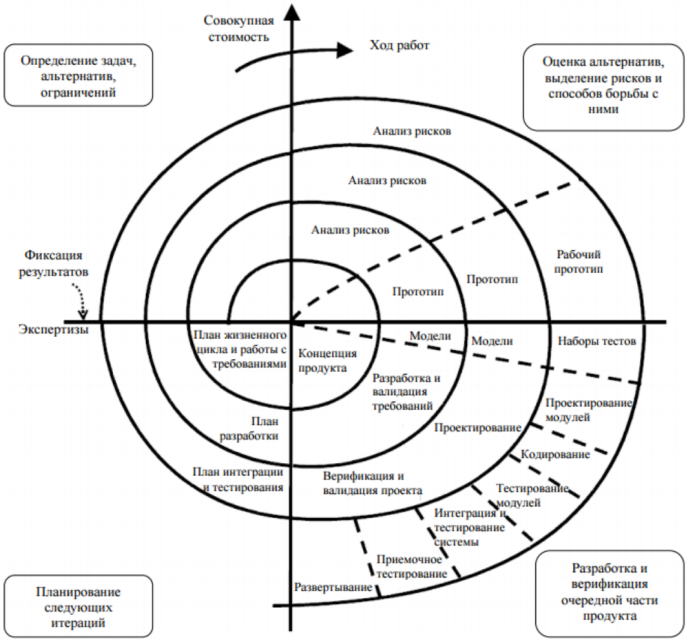
Среди разработчиков и исследователей, имевших дело с разработкой сложного ПО, практически с самого зарождения индустрии производства программ (см., например, статью 1968 года https://www.researchgate.net/publication/221329933_Iterative_Multi-Level_Modeling_-_A_Methodology_for_Computer_System_Design (дата обращения: 25.02.2023)) большую популярность имели модели эволюционных или итеративных процессов, поскольку они обладают большей гибкостью и способностью работать в меняющемся окружении. Итеративные или инкрементальные модели (известно несколько таких моделей) предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части. На первой итерации разрабатывается кусок системы, не зависящий от других. При этом большая часть или даже полный цикл работ проходится на нем, затем оцениваются результаты и на следующей итерации либо первый кусок переделывается, либо разрабатывается следующий, который может зависеть от первого, либо как-то совмещается доработка первого куска с добавлением новых функций. В результате на каждой итерации можно анализировать промежуточные результаты работ и реакцию на них всех заинтересованных лиц, включая пользователей, и вносить корректирующие изменения на следующих итерациях. Каждая итерация может содержать полный набор видов деятельности от анализа требований, до ввода в эксплуатацию очередной части ПО.



Итеративный процесс предполагает, что разные виды деятельности не привязаны на-

мертво к определенным этапам разработки, а выполняются по мере необходимости, иногда повторяются, до тех пор, пока не будет получен нужный результат. Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.

Развитием идеи итераций является спиральная модель жизненного цикла ПО, предложенная Боэмом в 1988 году[2]. Она предлагает каждую итерацию начинать с выделения целей и планирования очередной итерации, определения основных альтернатив и ограничений при ее выполнении, их оценки, а также оценки возникающих рисков и определения способов избавления от них, а заканчивать итерацию оценкой результатов проведенных в ее рамках работ. Основным ее новым элементом является общая структура действий на каждой итерации — планирование, определение задач, ограничений и вариантов решений, оценка предложенных решений и рисков, выполнение основных работ итерации и оценка их результатов.



Название «спиральной» эта модель получила из-за изображения хода работ в «полярных координатах», в которых угол соответствует выполняемому этапу в рамках общей структуры итераций, а удаление от начала координат — затраченным ресурсам. Каждый виток спирали соответствует созданию фрагмента или версии ПО. Дополнительное преимущество: на каждом витке спирали можно собрать метрические характеристики проекта (трудоемкость затрат, затраты на проект, длительность, документированность). Таким образом уточняется план-график дальнейшей работы.

[2] http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf (дата обращения: 25.02.2023).

Заметим, что спиральная модель может применяться на довольно высоком уровне абстракции: например, за квадрантом конструирования может скрываться целый жизненный цикл производства ПО новой версии.

Для применения этой модели может быть несколько причин:

- необходимость предупреждения рисков и быстрого реагирования на них;
- необходимость предоставлять заказчику частичные версии проектов для получения отзывов и пожеланий.

Минусы:

- требуется более искусное управление;
- трудности в определении момента перехода на следующий этап;
- необходимость поддержки целостности архитектуры;
- слишком большое количество витков потребует увеличения затрат на вспомогательную работу (планирование, рефакторинг и т.п.).

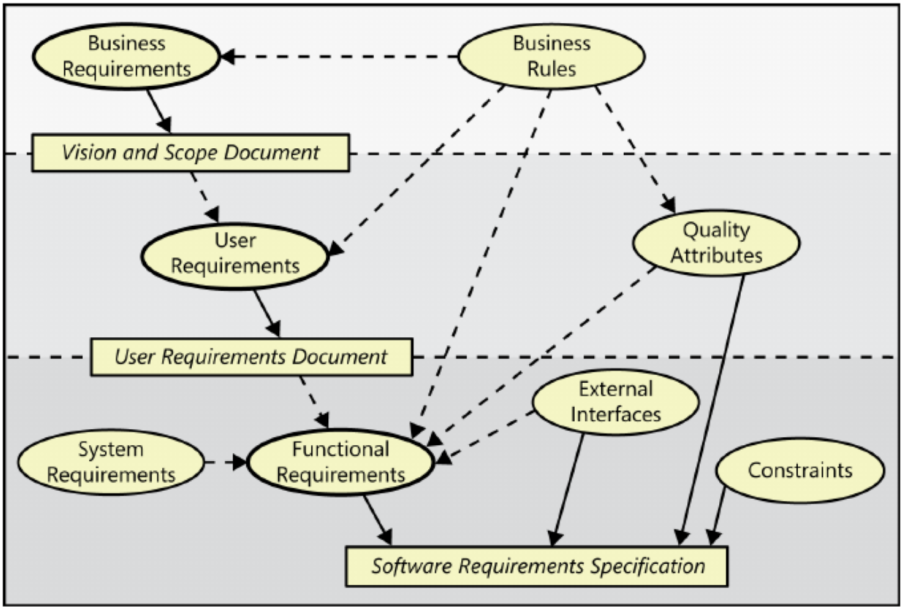
В отличие от водопадной модели, в спиральной нет предопределенного и обязательного набора витков. Каждый виток может стать последним при разработке системы, при его завершении составляются планы следующего витка. Наконец, виток является именно фазой, а не видом деятельности, как в водопадной модели, в его рамках может осуществляться много различных видов деятельности, то есть модель является двумерной.

2. Требования

Требование — это любое условие, которому должна соответствовать разрабатываемая система или программное средство. Требованием может быть часть функциональности, которой система должна обладать, или ограничение, которому система должна удовлетворять.

2.1. Типы требований

Требования к ПО состоят из нескольких уровней — бизнес-требования, требования пользователей и функциональные требования. Вдобавок каждая система имеет свои нефункциональные требования.



Бизнес-требования (business requirements) содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга и т.п. В этом документе объясняется, почему организации нужна такая система, то есть описаны цели, которые организация намерена достичь с её помощью.

Требования пользователей (user requirements) описывают цели и задачи, которые система позволит решить пользователям. К способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие-отклик». Таким образом, в этом документе указано, что клиенты смогут делать с помощью системы.

Функциональные требования (functional requirements) определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. Иногда именуемые требованиями поведения (behavioral requirements), они содержат положения с традиционным «должен» или «должна»: «Система должна по электронной почте отправлять пользователю подтверждение о заказе». Функциональные требования описывают, что разработчику необходимо реализовать.

Бизнес-правила (business rules) включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения, определяя, кто может выполнять конкретные варианты использования, или диктовать, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности. В этом случае можно отследить происхождение конкретных функциональных требований вплоть до соответствующих им бизнес-правил.

Функциональные требования документируются в спецификации требований к ПО

(software requirements specification, SRS), где описывается так полно, как необходимо, ожидаемое поведение системы. Спецификация может быть просто текстовым документом, базой данных или большой таблицей с требованиями, хранилищем данных в коммерческом инструменте управления требованиями или даже, может быть, просто кучей карточек для небольшого проекта. Спецификация требований к ПО используется при разработке, тестировании, обеспечении качества продукта, управлении проектом и связанных с проектом задачах.

В дополнение к функциональным требованиям спецификация содержит нефункциональные, где описаны цели и атрибуты качества. Атрибуты качества (quality attributes) представляют собой дополнительное описание функций продукта, выраженное через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся легкость и простота использования, легкость перемещения, целостность, эффективность и устойчивость к сбоям. Другие нефункциональные требования описывают внешние взаимодействия между системой и внешним миром, а также ограничения дизайна и реализации. Ограничения (constraints) касаются выбора возможности разработки внешнего вида и структуры продукта.

Так, менеджеры и сотрудники отдела маркетинга определяют бизнес-требования для ПО, которые помогут их компании работать эффективнее (для информационных систем) или успешно конкурировать на рынке (для коммерческих продуктов). Каждое требование пользователя должно быть сопоставлено бизнес-требованию. На основе требований пользователя аналитики определяют функции, которые позволят пользователям выполнять их задачи. Разработчикам необходимы функциональные и нефункциональные требования, чтобы создавать решения с желаемой функциональностью, определенным качеством и требуемыми рабочими характеристиками, не выходя за рамки налагаемых ограничений.

Для примера рассмотрим программу подготовки текстов. Бизнес-требование может выглядеть так: «Продукт позволит пользователям эффективно исправлять орфографические ошибки в тексте». Соответствующие требования пользователей могут содержать задачи (варианты использования) вроде такой: «Найти орфографическую ошибку» или «Добавить слово в общий словарь». Проверка грамматики имеет множество индивидуальных функциональных требований, которые связаны с такими операциями, как поиск и выделение слова с ошибкой, отображение диалогового окна с фрагментом текста, где это слово находится, и замена слова с ошибкой корректным вариантом по всему тексту. Атрибут качества «легкость и простота использования» (usability) заявляется как важный посредством слова «эффективно» в бизнес-требованиях.

3. Характеристики хороших требований

Хорошее требование должно обладать следующими характеристиками:

- Единичность — требование описывает одну и только одну вещь.
- Завершенность — требование полностью определено в одном месте и вся необходимая информация присутствует.
- Непротиворечивость — требование не противоречит другим требованиям и полностью соответствует документации.

- Атомарность — требование нельзя разделить на более мелкие.
- Отслеживаемость — требование полностью или частично соответствует деловым нуждам как заявлено заинтересованными лицами и это чётко задокументировано.
- Актуальность — требование не стало устаревшим с течением времени.
- Выполнимость — требование может быть реализовано в рамках проекта.
- Недвусмысленность — требование определено без обращения к техническому жаргону, акронимам и другим скрытым формулировкам. Оно выражает объекты и факты, а не субъективные мнения. Возможна одна и только одна его интерпретация. Определение не содержит нечетких фраз, использование отрицательных и составных утверждений запрещено.
- Обязательность — требование представляет собой определенную заинтересованным лицом характеристику, отсутствие которой ведет к неполноценности решения, которая не может быть проигнорирована. Необязательное требование — противоречие самому понятию требования.
- Проверяемость — реализованность требования может быть проверена.

3.1. Работа с требованиями

Работа с требованиями — процесс, включающий идентификацию, выявление, документацию, анализ, отслеживание, приоритизацию требований, достижение соглашений по требованиям и затем управление изменениями и уведомление заинтересованных лиц. Это непрерывный процесс на протяжении всего жизненного цикла продукта. В англоязычной среде также говорят о дисциплине «инженерия требований» (англ. Requirements Engineering). В процессе сбора требований важно принимать во внимание возможные противоречия требований различных заинтересованных лиц, таких как заказчики, разработчики или пользователи. Полнота и качество анализа требований играют ключевую роль в успехе всего проекта. А грамотное управление требованиями позволяет проекту избегать деградации при изменении требований в ходе разработки.

3.2. Выявление требований

Ранее мы обсуждали разные типы требований. Они собираются из разных источников на различных этапах работы над проектом, имеют различные цели и аудиторию и должны документироваться по-разному. Бизнес-требования не должны отменять каких-либо значительных пользовательских требований, кроме того, необходимо проследить, как из конкретных требований пользователей проистекают все функциональные требования. Также следует выявить нефункциональные характеристики, например предполагаемые качество и производительность. Рассмотрим основные моменты, про которые стоит помнить на этом этапе.

Определение процесса формулирования требований. Задокументируйте этапы выявления, анализа, определения и проверки требований. Наличие инструкций по выполнению ключевых операций поможет аналитикам качественно и согласованно выполнить их

работу. Кроме того, вам будет проще поставить задачи по созданию требований и графики, а также продумать необходимые ресурсы.

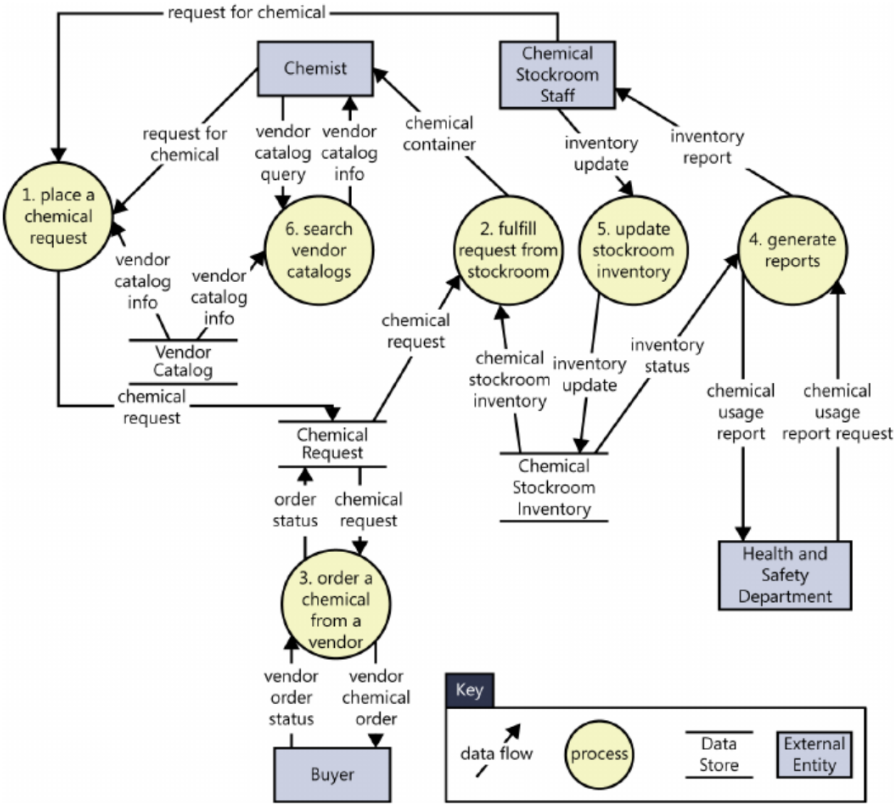
Определение классов пользователей и их характеристик. Чтобы не упустить из виду потребности отдельных пользователей, выделите их в группы. Например, это можно сделать по частоте работы с ПО, используемым функциям, уровню привилегий или навыкам работы. Опишите основных персонажей: их обязанности, необходимости, пожелания, личные характеристики и т.п. — всё, что способно повлиять на архитектуру продукта.

Выбор типичного пользователя в каждом классе пользователей. Очень здорово, если в каждой группе пользователей можно выделить человека, который сможет точно передавать настроения и нужды остальных в этой группе. Он представляет потребности определенного класса пользователей и принимает решения от их лица. В случае, когда все пользователи ваши коллеги, такого человека выбрать проще. При коммерческой разработке расспросите клиентов или используйте площадки бета-тестирования. Выбранные вами люди должны принимать постоянное участие в проекте и обладать полномочиями для принятия решений, касающихся пользовательских требований. Но нужно соблюдать осторожность и не позволять этому человеку выдавать собственные пожелания за требования всей группы.

Создание фокус-групп и проведение интервью с типичными пользователями. Фокус-группы особенно значимы при разработке коммерческих продуктов, когда приходится иметь дело с большой и разнородной клиентской базой. У фокус-групп обычно нет полномочий на принятие решений, однако мнение людей послушать полезно. С представителями групп пользователей можно провести персональные интервью.

Наблюдение за пользователями на рабочих местах. Наблюдая за работой пользователей, можно собрать много информации о контексте потенциального применения нового продукта. Простые диаграммы рабочих потоков, а также диаграммы потоков данных позволяют выяснить, где, как и какие данные задействовал пользователь. Документируя ход бизнес-процесса, удается определить требования к системе, предназначенной для поддержки этого процесса. Иногда даже выясняется, что для выполнения деловых задач клиентам вовсе и не требуется новое ПО.

Примером диаграммы, с помощью которой можно фиксировать рабочие процессы, является диаграмма потока данных, представленная ниже.



Изучение отчетов о проблемах работающих систем с целью поиска новых идей. Поступающие от клиентов отчеты о проблемах и предложения о расширении функциональности — отличный источник идей о возможностях, которые можно реализовать в следующей версии или новом продукте. Этой информации обычно бывает в избытке у сотрудников службы поддержки. Также можно проанализировать багтрекер и другие похожие системы.

Определение системных событий и реакции на них. Определите возможные внешние события и ожидаемую реакцию системы на них. Это могут быть сигналы и данные, получаемые от внешнего оборудования, а также временные события, вызывающие ответную реакцию, например ежедневная передача данных, генерируемых системой, внешнему объекту. В бизнес-приложениях бизнес-события напрямую связаны с задачами.

Создание документа об образе и границах проекта. Документ об образе и границах проекта содержит бизнес-требования к продукту. Описание образа проекта позволит всем заинтересованным лицам в общих чертах понять назначение продукта. Границы проекта определяют, что следует реализовать в первых версиях продукта. Образ и границы проекта — хорошая база для оценки предлагаемых требований. Образ продукта должен оставаться от версии к версии относительно стабильным, но для каждой глобальной версии необходимо составлять отдельный документ о границах.

3.3. Анализ требований

Анализ требований подразумевает их детализацию, гарантирующую, что требования понимают все заинтересованные лица, а также тщательное исследование требований на предмет ошибок, пробелов и других недостатков. Кроме того, анализ включает создание прототипов, анализ осуществимости и согласование приоритетов. Цель анализа — достаточно качественно и подробно описать требования, позволяющие менеджерам реалистично оценить все затраты на проект, а техническому персоналу — начать проектирование, сборку и тестирование.

Зачастую отдельные требования стоит представить несколькими способами, например в текстовой и графической форме. Это позволит выявить их особенности и проблемы, не заметные при представлении одним способом. Также это помогает всем заинтересованным лицам выработать согласованное представление об итогах разработки продукта.

Рассмотрим основные виды деятельности на этом этапе.

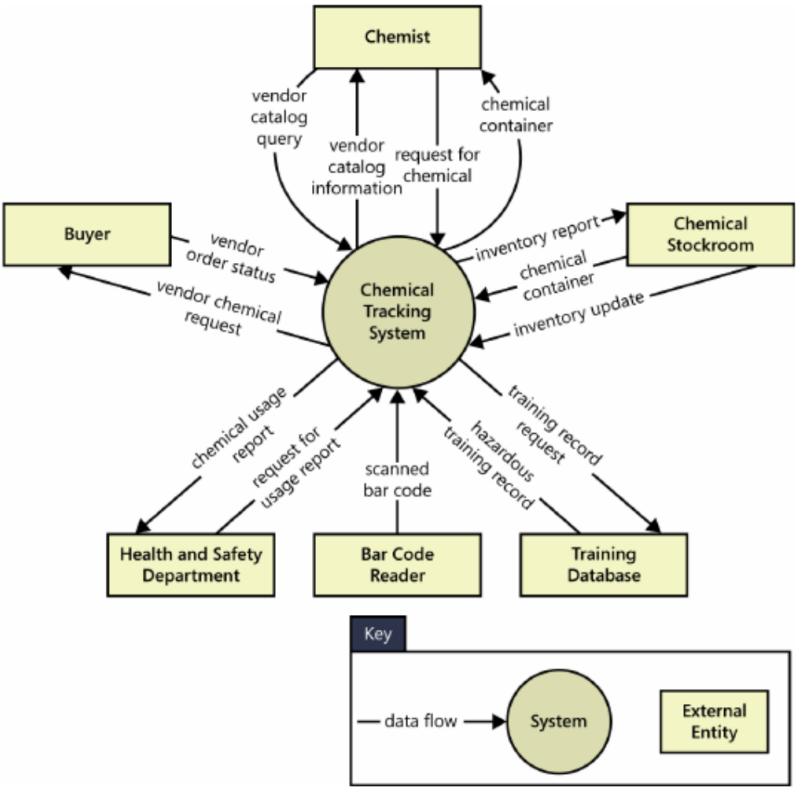
Анализ осуществимости требований. Проанализируйте, насколько реально реализовать каждое требование при разумных затратах и с приемлемой производительностью в предполагаемой среде. Рассмотрите риски, связанные с реализацией каждого требования, включая конфликты с другими требованиями, зависимость от внешних факторов и препятствия технического характера.

Определение приоритетов требований. Определите относительные приоритеты реализации функций продукта, решаемых задач или отдельных требований. На основании приоритетов установите, в какой версии будет реализована та или иная функция или набор требований. Подтверждая изменения, распределите все их по конкретным версиям и включите в план выпуска этих версий затраты, необходимые на внесение изменений. В ходе работы над проектом стоит периодически переоценивать и корректировать приоритеты в соответствии с потребностями клиента, условиями рынка и бизнес-целями.

Моделирование требований. В отличие от подробной информации, представленной в спецификации требований к ПО или пользовательского интерфейса прототипа, графическая модель анализа отображает требования на высоком уровне абстракции. Модели позволяют выявить некорректные, несогласованные, отсутствующие и избыточные требования. К таким моделям относятся диаграммы потоков данных, диаграммы «сущность-связь», диаграммы конечных автоматов, карты экранов, диаграммы классов, диаграммы последовательностей, диаграммы взаимодействий, таблицы и деревья решений и многое другое.

Создание словаря терминов. В этом документе соберите определения всех элементов и структур данных, связанных с системой, что позволяет всем участникам проекта использовать согласованные определения данных. На стадии работы над требованиями словарь должен содержать определения элементов данных, относящихся к предметной области, чтобы клиентам и разработчикам было проще общаться.

Создание контекстной диаграммы. Контекстная диаграмма — простая модель анализа, отображающая место новой системы в соответствующей среде. Она определяет границы и интерфейсы между разрабатываемой системой и сущностями, внешними для этой системы, например пользователями, устройствами и прочими информационными системами.



Создание пользовательского интерфейса и технических прототипов. Если разработчики или пользователи не совсем уверены насчет требований, создайте прототип — частичную, возможную или предварительную версию продукта, которая сделает концепции и возможности более осязаемыми. Оценка прототипа поможет всем заинтересованным лицам достичь взаимопонимания по решаемой проблеме.

3.4. Проверка требований

Процесс проверки гарантирует, что все положения требований корректны, отражают желаемые качественные характеристики и удовлетворяют потребностям клиента. Может оказаться, что требования, в спецификации требований к ПО выглядевшие превосходно, при реализации чреваты проблемами. В большинстве случаев удается выявить двусмысленности и неопределенности, написав для требований сценарии тестирования. Так как в большинстве случаев требования должны стать надежной основой для проектирования и итоговой проверки системы посредством системного тестирования или тестирования на приемлемость для пользователей, эти проблемы необходимо устранить.

Изучение документов с требованиями. Официальная проверка документирования требований — один из наиболее ценных способов проверки качества ПО. Соберите небольшую команду, члены которой представляют различные направления (например, аналитик, клиент, разработчик и специалист по тестированию), и тщательно изучите спецификацию

требований к ПО, модель анализа и соответствующую информацию на предмет недостатков. Также полезно провести в ходе формулирования требований их неофициальный предварительный просмотр. И хотя реализовать это на практике непросто, данный прием — один из самых ценных.

Определение критериев приемлемости. Предложите пользователям описать, как они собираются определять соответствие продукта их потребностям и его пригодность к работе.

Тестирование требований. На основе пользовательских требований создайте сценарии функционального тестирования и задокументируйте ожидаемое поведение продукта в конкретных условиях. Совместно с клиентами изучите сценарии тестирования и убедитесь, что они отражают нужное поведение системы. Проследите связь сценариев тестирования с функциональными требованиями и удостоверьтесь, что ни одно требование не пропущено и что для всех требований есть соответствующие сценарии тестирования. Если используемые формализмы позволяют, запустите сценарии, чтобы удостовериться в правильности моделей анализа и прототипов.

3.5. Основные риски, связанные с требованиями

Недостаточное вовлечение пользователей. Заказчики зачастую не понимают, почему так важно тщательно собрать требования и обеспечить их качество. Разработчики не всегда придают значение вовлечению пользователей в процесс из-за того, что среди них больше фанатов написания кода, чем любителей возиться с клиентами. Или же потому, что они считают, что всё уже знают о потребностях пользователей. В любом случае трудно добраться до людей, которые непосредственно будут иметь дело с продуктом, а те, кто выражает мнение пользователей, не всегда понимают, что тем нужно в реальности. Недостаточное вовлечение пользователей ведет к обнаружению ошибок в требованиях на поздних стадиях проекта, а значит, к задержке завершения проекта. Непосредственное и полноценное сотрудничество разработчиков с реальными пользователями на протяжении всего проекта очень сложно чем-то заменить.

Игнорирование классов пользователей. Большинство продуктов предназначены для нескольких групп пользователей, которые могут применять различные наборы функций с разной частотой и имеют опыт работы с ПО самого широкого диапазона. Если вы не определили важные классы пользователей для вашего продукта заранее, некоторые потребности клиентов могут быть не учтены. После идентификации всех классов удостоверьтесь, что по каждому из них у вас есть нужная информация.

Разрастание требований пользователей. Первоначально принятые планы не всегда основаны на реалистичном понимании размера и сложности требований, жесткая «заморозка» требований при этом не всегда возможна, а бездумное добавление требований в проект по ходу его развития может повлечь за собой серьёзные проблемы. Эти проблемы частично кроются в частых запросах пользователей на изменения, а частично в том, какими способами разработчики отвечают на них.

Чтобы управлять границами требований, для начала уточните бизнес-цели проекта, его рамки, ограничения, критерии успеха и ожидаемую пользу. Оцените, как предполагаемые характеристики или изменения требований отразятся на связанной с ними структуре. Эффективный процесс модификации заставляет интенсивнее работать аналитиков, которые помогут всем заинтересованным лицам принять обдуманные бизнес-решения относительно того, какие изменения следует принять, и увязать их стоимость со временем, ресурсами

или возможными компромиссами. Изменения зачастую критически важны для успеха, однако они всегда имеют цену.

По мере того, как продукт модифицируется в процессе разработки, его архитектура может медленно разрушаться. Вставляемые в код неаккуратные быстрые правки («костыли») затрудняют понимание и поддержку продукта. Чтобы минимизировать возможность потери качества продукта в результате проблем такого рода, вначале продумайте (или даже протестируйте по возможности), как возможные изменения отразятся на архитектуре и дизайне, а уж затем реализуйте их непосредственно в коде.

Двусмысленность требований — страшилка любой спецификации требований. Один из ее симптомов — пользователь имеет возможность интерпретировать одно и то же положение по-разному. Другой — что у нескольких читателей требований возникает разное представление о продукте. Кроме того, двусмысленность зачастую проистекает из неточности и плохой детализации требований, в результате чего разработчикам приходится заполнять возникающие пробелы собственными силами.

Двусмысленность ведет и к формированию различных ожиданий у заинтересованных лиц. Впоследствии некоторые из них удивляются, что же такое у нас получилось. Разработчики же впустую тратят время, занимаясь не теми проблемами, а тестировщики готовятся к проверке не тех особенностей поведения системы.

Один из способов обнаруживать двусмысленности — пригласить различных представителей пользователей для официальной экспертизы требований. Пусть они просмотрят документ и выскажут свои замечания. Если они интерпретируют требования различными способами, то пусть неясность лучше проявится сейчас, а не гораздо позже, когда исправлять её будет сильно дороже. Другой способ вскрыть двусмысленность — написать вариант тестирования для требования или построить прототип.

Излишняя дополнительная функциональность. Под этим пунктом понимают такие ситуации, когда разработчики добавляют функции, которых нет в спецификации, но им кажется, что это понравится пользователям. Зачастую же клиентам не нужны такие избыточные возможности, получается, что время, отведенное на реализацию, тратится впустую. Прежде чем просто вставлять новые функции, разработчики и аналитики должны представить свои творческие идеи на суд заказчиков. Задача команды — чётко соблюдать требования спецификации, а не действовать за спиной клиентов без одобрения.

Другая сторона проблемы — пользователи иногда требуют функции или элементы интерфейса, которые выглядят отлично, но не представляют особой ценности для продукта. Всё, что вы захотите добавить, стоит времени и денег, поэтому постарайтесь осознать ценность своевременного выпуска продукта. Чтобы избежать такой ситуации, отслеживайте каждый кусок функциональности до его первоисточника, чтобы четко понимать, почему именно он включен в продукт. Применение вариантов использования для извлечения требований поможет сосредоточиться на выборе тех элементов, которые помогут пользователям выполнять их бизнес-задачи.

Урезанная спецификация. Иногда сотрудников отдела маркетинга или менеджеров охватывает искушение создать урезанный вариант спецификации. Они ожидают, что разработчики «нарастят мясо» на основе этих набросков, пока проект развивается. Это годится для тесно сплочённой команды, которая занимается небольшой системой, или когда выполняется проект-исследование, или когда требования действительно гибкие. Хотя в большинстве случаев это всё-таки раздражает разработчиков (которые могут действовать при некорректных предположениях и с ограниченными инструкциями) и дезориентирует

клиентов (которые не получат тот продукт, который они воображали).

Небрежное планирование. Когда к вам подбегает менеджер и говорит «Я кое-что придумал для нового продукта. Когда вы сможете это сделать?», не отвечайте на подобный вопрос, пока больше не узнаете о проблеме. Неопределенные, недетализированные требования порождают слишком оптимистические оценки, за которыми часто следует перерасход времени и денег. При плохо просчитанной смете проект больше всего страдает из-за затрат на частые изменения требований, пропущенных требований, недостаточного взаимодействия с пользователями, недетализированной спецификации требований и плохого анализа. Когда вы высказываете оценку, то указывайте лучше временные рамки: лучший вариант, наиболее вероятный и худший вариант (хотя это в жизни на самом деле тоже не помогает, об оценках мы будем говорить в курсе дальше).

3.6. Создаваемые документы

На этапе спецификации требований обычно создаются следующие документы:

- документ об образе и границах системы,
- глоссарий,
- модель требований,
- прототип пользовательского интерфейса,
- спецификация требований.

Создавать их можно с разной степенью формальности или даже не создавать совсем, но наличие их может сильно помочь при дальнейшем ходе проекта. Глоссарий и прототипы пользовательских интерфейсов мы уже в некотором виде обсудили, поговорим про остальные документы.

3.7. Документ об образе и границах системы

Если у участников проекта разное представление о предполагаемых границах и целях проекта, то им трудно будет договориться о том, какое функциональное требование относится к спецификации требований, а какое не относится.

Проект, в котором нет четко определенного и согласованного направления, сложно довести до успешного завершения, поскольку никто не понимает, где оно. Участники проекта могут, сами того не осознавая, решать прямо противоположные задачи, если у них разные бизнес-цели и приоритеты. Четкое представление образа и границ проекта особенно важно при распределенной разработке ПО, когда географическое разделение участников замедляет ежедневное взаимодействие, упрощающее коллективную работу.

Проблемы, касающиеся образа и границ, необходимо разрешать до спецификации детализированных функциональных требований. Положение о рамках и ограничениях проекта необычайно полезно при обсуждении предлагаемых функций и целевых выпусков. Кроме того, на него можно ссылаться при принятии решений об изменении и расширении

требований. В некоторых компаниях основные положения об образе и границах помещают на схему, которую приносят на каждое проектное совещание, чтобы все смогли быстро оценить, не выходит ли предложенное изменение за рамки проекта.

Образ или концепция продукта (product vision) выстраивает работу всех заинтересованных лиц в одном направлении. Он описывает, что продукт представляет собой сейчас и каким он станет впоследствии. Границы проекта (project scope) показывают, к какой области конечного долгосрочного образа продукта будет направлен текущий проект. В положении о границах определена черта между тем, что входит в проект и тем, что остается вовне. То есть указанные рамки также определяют ограничения проекта.

Рассмотрим возможную структуру такого документа.

1. **Бизнес-требования.** Проекты должны выпускаться с убеждением, что новый продукт для кого-то сделает мир лучше. Бизнес-требования описывают основные преимущества, которые новая система даст ее заказчикам, покупателям и пользователям.

   (a) **Исходные данные.** Этот раздел суммирует обоснование и содержание нового продукта. В него помещают общее описание предыстории или ситуации, в результате которых было принято решение о создании продукта.

   (b) **Возможности бизнеса.** Для коммерческого продукта описывают существующие рыночные возможности и рынок, на котором продукту придется конкурировать с другими продуктами. Обычно здесь описывают бизнес-проблему, которая разрешается посредством этого продукта, или бизнес-процессы, для улучшения которых требуется продукт, а также среду, в которой система будет использоваться. Кроме того, сюда можно же добавить и сравнительную оценку существующих продуктов и возможных решений, указав, в чем заключается привлекательность продукта и его преимущества. Покажите, насколько он соответствует тенденциям рынка, развитию технологий или корпоративной стратегии.

   (c) **Бизнес-цели и критерии успеха.** Данный раздел суммирует важные преимущества бизнеса, предоставляемые продуктом, в количественном и измеряемом виде. Здесь же нужно отметить, как заинтересованные лица будут определять и измерять успех проекта. Установите факторы, которые максимально влияют на успех проекта, и те, которые организация может контролировать, и те, которые находятся вне сферы ее влияния. Определите меру для оценки того, были ли достигнуты бизнес-цели.

   (d) **Потребности клиентов или рынка.** Опишите потребности типичных покупателей или целевых сегментов рынка, включая потребности, которые не удовлетворяют настоящие продукты или информационные системы. Представьте проблемы, с которыми в настоящее время сталкиваются клиенты и которые решит новая система, и предоставьте примеры того, как покупатели будут использовать этот продукт. Определите на высоком уровне все известные важные требования к интерфейсу или производительности, но не касайтесь деталей дизайна или реализации.

   (e) **Бизнес-риски.** Этот раздел обобщает важнейшие бизнес-риски, связанные с созданием этого продукта. В категории рисков входят рыночная конкуренция,

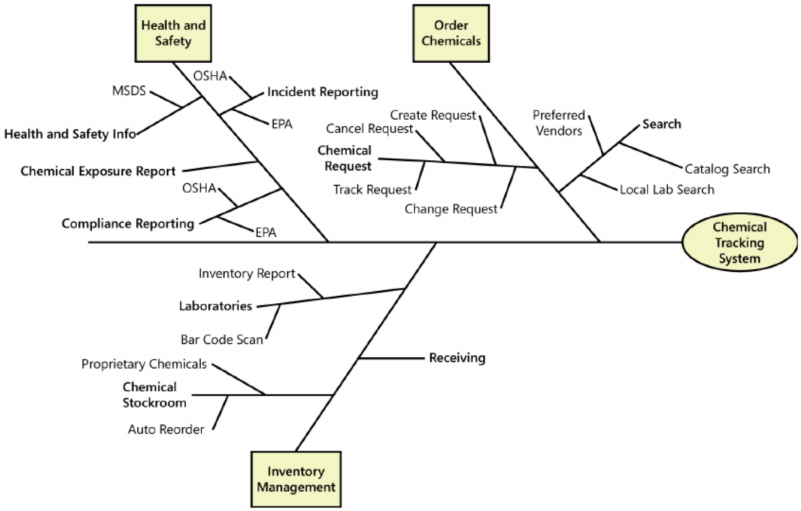
временные факторы, приемлемость для пользователей, проблемы, связанные с реализацией, и возможные негативные факторы, влияющие на бизнес. Оцените возможные потери от каждого фактора риска, вероятность его возникновения и вашу способность контролировать его. Определите все возможные действия по смягчению ситуации.

2. **Концепция (vision) решения.** В этом разделе документа определяется стратегический образ системы, позволяющей выполнять бизнес-задачи. Этот образ обеспечивает основу для принятия решений в течение жизненного цикла продукта. В него не надо включать детали функциональных требований или информацию, связанную с планированием проекта.

   (a) **Положение об образе проекта.** Составьте сжатое положение об образе проекта, обобщающее долгосрочные цели и назначение нового продукта. В этом документе следует отразить сбалансированный образ, удовлетворяющий различные заинтересованные лица. Он может быть несколько идеалистичным, но должен быть основан на существующих или предполагаемых рыночных факторах, архитектуре предприятия, стратегическом направлении развития компании или ограничениях ресурсов. Далее показан шаблон, состоящий из ключевых слов, который неплохо подходит для этого раздела:

   Для [целевая аудитория покупателей], которые [потребности или возможности пользователей], [имя продукта] является [категория продукта], который [ключевое преимущество, основная причина для покупки или использования]. В отличие от [основной конкурирующий продукт, текущая система или текущий бизнес-процесс] наш продукт [основное отличие и преимущество нового продукта].

   (b) **Основные функции.** Опишите каждую основную функцию нового продукта или возможность, предоставляемую пользователям, подчеркивая те из них, которые отличают его от предыдущих или конкурирующих продуктов. Дав каждой функции уникальное имя, вы сможете отследить каждую функцию до отдельных требований пользователей, функциональных требований и других элементов систем. Для наглядного отображения декомпозиции функций проекта часто удобно использовать следующую диаграмму:



Она отображает функции, организованные в иерархически устроенные логические группы. Обычно на таких диаграммах отображают не больше трёх уровней декомпозиции.

(c) **Предположения и зависимости.** Задокументируйте все предположения, сделанные заинтересованными лицами, когда они обдумывали проект и создавали данный документ об образе и границах. Часто предположения одних лиц не разделяют другие стороны. Если вы запишите их и просмотрите позже, то получите возможность обговорить основные положения проекта. Так вы избежите путаницы и ухудшения ситуации в будущем. Также задокументируйте важнейшие зависимости проекта от внешних факторов: изменения индустриальных стандартов или правительственных положений, других проектов, поставщиков со стороны или партнеров по разработке.

3. **Масштабы и ограничения проекта.** В этом разделе необходимо указать, что может делать система, а что не может. Рамки и ограничения помогают установить реалистичные ожидания заинтересованных лиц. Иногда клиенты запрашивают функции, слишком дорогостоящие или выходящие за предполагаемые границы продукта. Требования, выходящие за границы продукта, следует отклонять, если только они не настолько ценны, чтобы специально под них расширить проект, естественно, соответствующим образом изменив в бюджет, график и кадровый состав. Документируйте отклоненные требования и причины отказа от них, поскольку они имеют свойство появляться снова.

(a) **Объем первоначальной версии.** Обобщает основные запланированные функции, включенные в первоначальную версию продукта. Опишите характеристики качества, которые позволят продукту предоставлять предполагаемые выгоды различным классам пользователей. Если ваша задача — сосредоточиться на разработке и уложиться в график, вам следует избегать искушения включить в версию 1.0 каждую функцию, которая когда-нибудь в будущем может понадобиться какому-то потенциальному покупателю.

Увеличение сроков и сдвиг графика — типичный исход такого расползания объема. Сосредоточьтесь на наиболее ценных функциях, имеющих максимально приемлемую стоимость, годных для самой широкой целевой аудитории, которые удастся создать как можно раньше. Версия 1.0 не обязательно должна быть супербыстрой, красиво оформленной или легкой в использовании, но она должна быть надежной. Первая версия системы выполняет лишь базовые задачи. В будущие выпуски будут включены дополнительные функции, возможности и средства, обеспечивающие легкость и простоту использования.

(b) **Объем последующих версий.** Если вы представляете поэтапную эволюцию продукта, укажите, какие функции будут отложены, и желательные сроки последующих выпусков. Чем дальше вы заглядываете, тем более расплывчатыми будут границы проекта.

(c) **Ограничения и исключения.** Определение границы между тем, что входит, и тем, что не входит в границы проекта, — отличный способ управления расползанием объёма и ожиданиями клиентов. Перечислите все возможности или характеристики, которых могут ожидать заинтересованные в проекте лица, но включение которых в продукт или в определенную версию не запланировано.

4. **Бизнес-контекст.** В этом разделе обобщаются некоторые бизнес-проблемы проекта, включая профили основных категорий заинтересованных лиц и приоритеты управления.

(a) **Профили заинтересованных лиц.** Заинтересованными в проекте лицами (stakeholders) называются отдельные лица, группы или организации, которые активно вовлечены в проект, на которых влияет результат проекта и которые сами могут влиять на этот результат. Если у вас уже созданы профили персонажей, можно поместить их сюда. Тут важно для каждого типа пользователей указать их вероятное отношение к продукту и основную ценность или преимущество, которое продукт принесет этой группе пользователей.

(b) **Приоритеты проекта.** Чтобы принимать эффективные решения, заинтересованные лица должны договориться о приоритетах проекта. Один из подходов к этому заключается в рассмотрении пяти измеряемых параметров проекта: функции (или объем), качество, график, затраты и кадры. В любом проекте каждый из этих параметров относится к одной из трех категорий:

- ограничение — лимитирующий фактор, в рамках которого должен оперировать менеджер проекта;
- ключевой фактор — важный фактор успеха, ограниченно гибкий при изменениях;
- степень свободы — фактор, который менеджер проекта может до определенной степени изменять и балансировать относительно других параметров.

Задача менеджера проекта — настроить те факторы, которые представляют собой степени свободы для достижения ключевых факторов успеха проекта в рамках, налагаемых ограничениями. Не все факторы могут быть ключевыми, как

и не все — ограничениями. Менеджеру проекта необходима определенная степень свободы для того, чтобы он мог реагировать должным образом на изменение требований к проекту или внешних обстоятельств. Представьте себе, что отдел маркетинга неожиданно требует создать продукт на месяц раньше срока. Какова будет ваша реакция? Вы отложите реализацию определенных требований до более поздней версии? Сократите запланированный цикл тестирования системы? Оплатите сверхурочную работу вашим специалистам или пригласите специалистов по контракту для ускорения разработки? Привлечёте ресурсы других проектов для разрешения ситуации? Именно от приоритетов проекта зависят ваши действия в подобных ситуациях, и очень хорошо, если эти приоритеты будут зафиксированы заранее.

(c) **Операционная среда.** Опишите среду, в которой будет использоваться система, и определите важнейшие требования к доступности, надежности, производительности и целостности. Эта информация существенно влияет на определение архитектуры системы, что является первым и часто самым важным этапом дизайна. Архитектура системы, предназначенной для поддержки пользователей, которые находятся далеко друг от друга и которым необходим круглосуточный доступ, сильно отличается от той, что предназначена для доступа пользователей, находящихся рядом, только в рабочие часы. На нефункциональные требования, такие как отказоустойчивость и способность обслуживать систему во время ее работы, требуется значительное количество средств, отпущенных на дизайн и реализацию.

3.8. Моделирование требований

В течение многих лет аналитики извлекали информацию о требованиях пользователей из сценариев использования. Сценарием называется описание одного варианта использования системы. Со временем на основе разработок Ивара Якобсона и других людей этот подход был преобразован в методику, где для сбора информации и моделирования требований применяются варианты использования. Последние прекрасно себя зарекомендовали для разработки широкого класса систем, активно взаимодействующих с пользователем и внешним миром. (Для продуктов, сложность работы которых заключается в выполняемых вычислениях или генерировании отчетов, а не во взаимодействии пользователя и системы, этот метод будет не так полезен, поскольку варианты использования там будут довольно простые и немногочисленные.)

Варианты использования меняют традиционный подход к сбору информации: пользователей не спрашивают, как прежде, что, с их точки зрения, должна делать система, а выясняют, какие задачи собирается с ее помощью решать пользователь. Цель такого подхода — описать все подобные задачи. До включения каждого варианта использования в утвержденную версию требований, заинтересованные в проекте лица проверяют, не выходит ли он за границы проекта. Теоретически в конечный набор вариантов использования должна входить вся желаемая функциональность системы.

Для моделирования вариантов использования в UML есть специальный вид диаграмм: диаграмма вариантов использования (use case diagram). В ней есть две основные сущности: роли/актёры и сценарии/варианты использования.

Актёр — это кто-то или что-то вне системы, влияющий на систему или находящийся под её влиянием. Актёр может быть человеком, устройством, другой системой или подсистемой. Человек в реальном мире может быть представлен несколькими актёрами, если у них есть несколько различных ролей и целей по отношению к системе. Они взаимодействуют с системой и производят над ней некоторые действия. Типовые примеры актёров:

- люди или программные системы, которым проектируемая система требуется для выполнения их задач;
- люди или программные системы, которые необходимы для осуществления проектируемой системой своих функций;
- люди или программные системы, взаимодействующие с внешними программными и аппаратными интерфейсами проектируемой системы;
- люди или программные системы, выполняющие вспомогательные функции администрирования и поддержки.

Сценарий или случай использования представляет собой действие или последовательность действий, выполняемых системой в ответ на некоторое внешнее событие (инициируемое актёром). Или, более концептуально, сценарий использования представляет цель пользователя системы и процедуру, выполняемую пользователем для достижения этой цели.

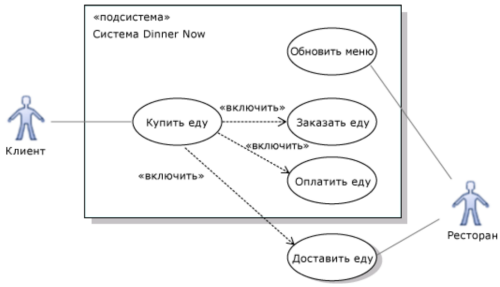
Диаграмма сценариев использования является самым общим представлением функциональных требований к системе и не показывает порядок выполнения шагов для достижения цели каждого из сценариев использования. В дальнейшем эти диаграммы могут раскрываться в описание алгоритмов или потока событий при помощи других диаграмм (например, диаграмм активностей) или текстом в других документах.

Например, система продажи продуктов питания через Интернет должна позволять клиентам выбирать элементы меню, а ресторанам-поставщикам — обновлять меню. Это можно объединить в схеме сценариев использования.

Также можно показать, как сценарий использования составляется из более мелких сценариев. Например, заказ продуктов питания — часть процесса покупки, который также включает оплату и доставку.

Кроме того, на этих диаграммах можно показать, какие сценарии использования включены в область разрабатываемых подсистем. Например, изображенная на рисунке ниже подсистема не входит в состав сценария использования «Доставка еды». Это помогает задать контекст для разработки. Это также можно использовать при обсуждении, что будет включено в последующие релизы. Например, можно обсудить, будет ли компонент «Оплата еды» в первоначальном релизе системы функционировать непосредственно между рестораном и клиентом (а не обрабатываться в системе). В этом случае в начальном выпуске

можно переместить компонент «Оплата еды» за пределы прямоугольника системы Dinner Now.



Если число сценариев использования слишком велико, то для упрощения читаемости и поддержки модели целесообразно разделить их по пакетам. Это также упрощает понимание модели и распределение ответственности путем назначения разработчиков, ответственных за пакеты сценариев использования. Пакеты позволяют организовать иерархию требований.

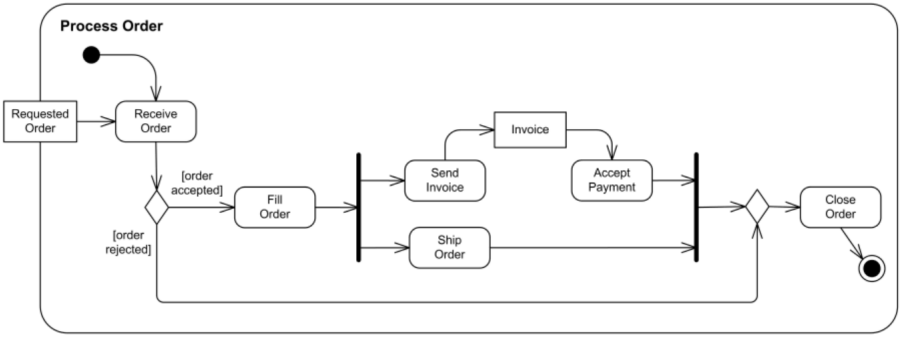
Сценариям могут быть назначены приоритеты: критический, важный и вспомогательный. Критичные сценарии представляют основную системную функциональность или имеют существенное архитектурное значение. Важные сценарии определяют такие системные функции, как сбор статистики, составление отчетов, контроль и функциональное тестирование. Если они не реализованы, то система может выполнять свое предназначение, но со значительно худшим качеством сервиса. Вспомогательные сценарии определяют дополнительную функциональность системы.

Более подробно про синтаксис диаграмм вариантов использования можно почитать, например, в книге Мартина Фаулера «UML: Основы».

Раскрываться сценарии с этих диаграмм могут по-разному, ниже приведён пример формата текстового описания.

ID and Name:	UC-4 Request a Chemical		
Created By:	Lori	Date Created:	8/22/13
Primary Actor:	Requester	Secondary Actors:	Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.		
Trigger:	Requester indicates that he wants to request a chemical.		
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.		
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.		
Normal Flow:	**4.0 Request a Chemical from the Chemical Stockroom** 1. Requester specifies the desired chemical. 2. System lists containers of the desired chemical that are in the chemical stockroom, if any. 3. System gives Requester the option to View Container History for any container. 4. Requester selects a specific container or asks to place a vendor order (see 4.1). 5. Requester enters other information to complete the request. 6. System stores the request and notifies the Chemical Stockroom.		
Alternative Flows:	**4.1 Request a Chemical from a Vendor** 1. Requester searches vendor catalogs for the chemical (see 4.1.E1). 2. System displays a list of vendors for the chemical with available container sizes, grades, and prices. 3. Requester selects a vendor, container size, grade, and number of containers. 4. Requester enters other information to complete the request. 5. System stores the request and notifies the Buyer.		
Exceptions:	**4.1.E1 Chemical Is Not Commercially Available** 1. System displays message: No vendors for that chemical. 2. System asks Requester if he wants to request another chemical (3a) or to exit (4a). 3a. Requester asks to request another chemical. 3b. System starts normal flow over. 4a. Requester asks to exit. 4b. System terminates use case.		
Priority:	High		
Frequency of Use:	Approximately 5 times per week by each chemist, 200 times per week by chemical stockroom staff		
Business Rules:	BR-28, BR-31		
Other Information:	The system must be able to import a chemical structure in the standard encoded form from any of the supported chemical drawing packages.		
Assumptions:	Imported chemical structures are assumed to be valid.		

Для более наглядного представления сценариев могут использоваться диаграммы активности UML. Их синтаксис сильно похож на всем известные блок-схемы, и такие диаграммы могут понимать даже самые далёкие от программирования специалисты.



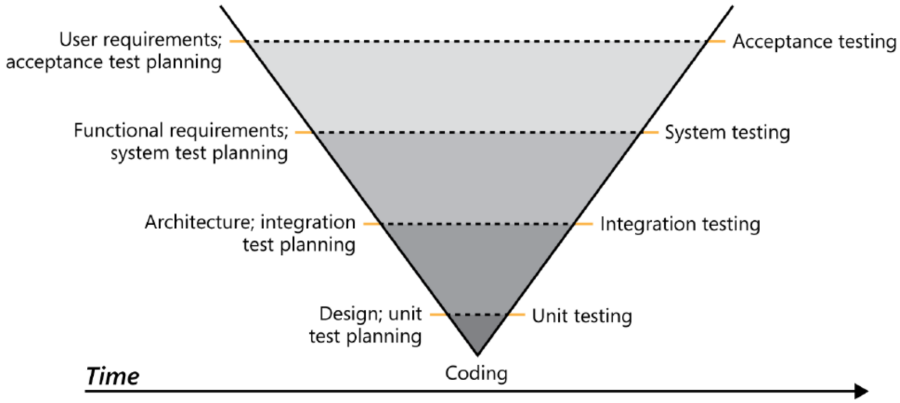
3.9. Спецификация требований к ПО

На основе созданных моделей и документов часто строится объемлющий документ, содержащий в себе всю собранную информацию. Обычно он прикладывается к договору/контракту или используется как основа для формирования технического задания. Формат такого документа может быть такой:

1. Введение
    (a) Цели
    (b) Соглашения о терминах
    (c) Предполагаемая аудитория
    (d) Масштаб проекта
    (e) Ссылки на источники
2. Общее описание
    (a) Видение продукта
    (b) Функциональность продукта
    (c) Классы и характеристики пользователей
    (d) Среда функционирования продукта
    (e) Рамки, ограничения, правила и стандарты
    (f) Документация для пользователей
    (g) Допущения и зависимости
3. Функциональность системы
    (a) Функциональный блок X
    (b) Описание и приоритет
    (c) Причинно-следственные связи, алгоритмы
    (d) Функциональные требования
4. Требования к внешним интерфейсам
    (a) Интерфейсы пользователя (UX)
    (b) Программные интерфейсы
    (c) Интерфейсы оборудования
    (d) Интерфейсы связи и коммуникации
5. Нефункциональные требования
    (a) Требования к производительности
    (b) Требования к сохранности (данных)
    (c) Критерии качества ПО
    (d) Требования к безопасности системы
6. Прочие требования

3.10. Требования и тестирование

Требования в явном виде являются источником информации для тестирования системы. На этапе проектирования требования можно использовать для проверки адекватности архитектуры, по ходу разработки требования могут превращаться в приёмочные тесты к системе в целом.



3.11. Управление требованиями

Начальные требования неизбежно корректируются в процессе работы клиентами, менеджерами, специалистами по маркетингу, разработчиками и другими лицами. Для эффективного управления требованиями необходим процесс, позволяющий предлагать изменения и оценивать их возможную стоимость и влияние на проект. Контроль за выполнением требований на разных стадиях разработки и тестирования системы позволяет лучше понять состояние проекта в целом. Рассмотрим основные виды деятельности, направленные на управление требованиями во время разработки проекта.

Определение процесса управления изменениями. Определите процесс представления, анализа и утверждения или отклонения изменений к требованиям. Применяйте его для управления всеми предлагаемыми изменениями без исключения. Из представителей заинтересованных в проекте лиц организуйте совет по управлению изменениями, который будет получать информацию о предполагаемых изменениях требований, оценивать ее, решать, какие изменения принять, а какие отклонить, и определять, в какой версии продукта будет внедрена та или иная функция. Даже если этот совет будет состоять из одного человека, важно, чтобы он был, и все изменения проходили через него. Если изменения будут попадать в проект мимо этого совета (например, напрямую от заказчиков к программистам), это по сути сведёт на нет все усилия по планированию и управлению ходом проекта.

Создание базовой версии и управление версиями требований. Базовая версия содержит требования, утвержденные для реализации в конкретной версии продукта. После определения базовых требований изменения можно вносить только в соответствии с процессом управления изменениями. Присвойте всем версиям спецификации требований уникальные идентификаторы, чтобы избежать путаницы между черновыми вариантами и базовыми версиями, а также между предыдущей и текущей версиями требований. Более

надежное решение — управлять версиями документов с требованиями при помощи соответствующих средств управления конфигурацией.

Использование средств версионирования и управления требованиями. Простейший механизм управления версиями — именовать вручную каждую версию спецификации требований к ПО согласно соглашению. Попытка различать версии документов по дате изменения или дате печати часто приводит к ошибкам и неразберихе, лучше на это не закладываться.

Более сложный метод предполагает сохранение спецификации требований в средстве контроля версий, по аналогии с тем, как версионируется программный код. Хранить файлы в формате Microsoft Word в git'е — не самое красивое решение, но оно вполне себе работает, и оно гораздо лучше, чем ничего. Наиболее надежный метод контроля версий заключается в хранении требований в базе данных коммерческого средства управления требованиями. Эти средства отслеживают полную историю изменений требований, что особенно важно, если нужно вернуться к более ранней версии требования. Такое средство позволяет вносить комментарии, где можно разумно обосновать решение о добавлении, изменении или удалении требования; эти комментарии могут оказаться полезными при необходимости обсуждения требования в будущем.

Вне зависимости от того, как именно происходит версионирование, для каждого требования полезно указывать следующие атрибуты:

- дата создания, автор;
- номер текущей ревизии;
- лицо, ответственное за удовлетворение требования;
- происхождение или источник требования;
- состояние (proposed, approved, implemented, verified, rejected и т.п.);
- приоритет;
- подсистемы, для которых актуально требование;
- номер версии продукта, для которой актуально требование;
- критерий приемлемости, используемый метод проверки.

Анализ влияния изменений требований. Анализ влияния изменений помогает совету по управлению изменениями принимать обоснованные решения. Оцените, как каждое предлагаемое изменение требований повлияет на проект. Выявите другие требования, элементы архитектуры, исходный код и сценарии тестирования, которые, возможно, придется изменить. Определите, что необходимо для реализации изменений, и оцените затраты на реализацию.

Оценка изменяемости требований. Регулярно фиксируйте количество требований, внесенных в базовую версию, а также число предложенных и одобренных изменений (добавлений, модификаций и удалений). Если требования формируются не самим клиентом, а от его лица, может оказаться, что проблема понята плохо, границы проекта определены нечетко, бизнес стремительно меняется, при сборе информации многие требования были упущены или внутрикорпоративные политики меняются в худшую сторону.

4. Литература

- Karl Wiegers. Software Requirements: https://www.amazon.com/Software-Requirements-Developer-Best-Practices/dp/0735679665
- Полезный курс на Степике: https://stepik.org/course/1128
- Блог на тему: http://foranalysts.blogspot.ru/