Управление IT-проектами и процессами

Лекция №7
Тема: Управление требованиями

Козбакова Айнур Холдасовна– PhD, профессор, кафедра «Информационные системы» Факультет «Инжиниринг и информационных технологии» 01.09.2025 г.

Виды и свойства требований

Разделим требования на две большие группы – функциональные и нефункциональные.

Функциональные требования являются детальным описанием поведения и *сервисов* системы, ее функционала. Они определяют то, что система должна уметь делать.

Нефункциональные требования не являются описанием функций системы. Этот вид требований описывает такие характеристики системы, как *надежность*, особенности поставки (наличие инсталлятора, документации), определенный уровень качества (например, для новой Java-машины это будет означать, что она удовлетворяет набору тестов, поддерживаемому компанией *Sun*). Сюда же могут относиться требования на средства и процесс разработки системы, требования к *переносимости*, соответствию стандартам и т.д. Требования этого вида часто относятся ко всей системе в целом. На практике, особенно начинающие специалисты, часто забывают про некоторые важные нефункциональные требования.

Сформулируем ряд важных свойств требований.

• *Ясность, недвусмысленность* — однозначность понимания *требований заказчиком* и разработчиками. Часто этого трудно достичь, поскольку конечная формализация требований, выполненная с точки зрения потребностей дальнейшей разработки, трудна для восприятия заказчиком или *специалистом предметной области*, которые должны проинспектировать правильность формализации.

• *Полнота и непротиворечивость*.

• *Необходимый уровень детализации*. Требования должны обладать ясно осознаваемым уровнем *детализации*, стилем описания, способом формализации: либо это описание свойств *предметной области*, для которой предназначается ПО, либо это *техническое задание*, которое прилагается к контракту, либо это проектная спецификация, которая должна быть уточнена в дальнейшем, при детальном проектировании. Либо это еще что-нибудь. Важно также ясно видеть и понимать тех, для кого данное описание требований предназначено, иначе не избежать недопонимания и последующих за этим трудностей. Ведь в разработке ПО задействовано много различных специалистов – инженеров, программистов, *тестировщиков*, представителей заказчика, возможно, будущих пользователей – и все они имеют разное образование, профессиональные навыки и специализацию, часто говорят на разных языках. Здесь также важно, чтобы требования были максимально абстрактны и независимы от реализации.

• *Прослеживаемость* — важно видеть то или иное требование в различных моделях, документах, наконец, в коде системы. А то часто возникают вопросы типа – "Кто знает, почему мы решили, что такой-то модуль должен работать следующим образом ….?". Прослеживаемость функциональных требований достигается путем их дробления на отдельные, элементарные требования, присвоение им *идентификаторов* и создание трассировочной модели, которая в идеале должна протягиваться до программного кода. Хочется например, знать, где нужно изменить код, если данное требование изменилось. На практике полная формальная прослеживаемость труднодостижима, поскольку логика и структура реализации системы могут сильно не совпадать с таковыми для модели требований. В итоге одно требование оказывается сильно "размазано" по коду, а тот или иной участок кода может влиять на много требований. Но стремиться к прослеживаемости необходимо, разумно совмещая формальные и неформальные подходы.

• *Тестируемость и проверяемость* — необходимо, чтобы существовали способы оттестировать и проверить данное требование. Причем, важны оба аспекта, поскольку часто проверить-то заказчик может, а вот тестировать данное требование очень трудно или невозможно в виду *ограниченности доступа* (например, по соображениям безопасности) к окружению системы для команды разработчика. Итак, необходимы процедуры проверки – выполнение тестов, проведение инспекций, проведение формальной *верификации* части

требований и пр. Нужно также определять "планку" качества (чем выше качество, тем оно дороже стоит!), а также критерии полноты проверок, чтобы выполняющие их и *руководители проекта* четко осознавали, что именно проверено, а что еще нет.

• *Модифицируемость*. Определяет процедуры внесения изменений в требования.

Варианты формализации требований

Вообще говоря, требования как таковые – это некоторая *абстракция*. В реальной практике они всегда существуют в виде какого-то представления – документа, модели, *формальной спецификации*, списка и т.д. Требования важны как таковые, потому что оседают в виде понимания разработчиками нужд заказчика и будущих пользователей создаваемой системы. Но так как в программном проекте много различных аспектов, видов *деятельности* и фаз разработки, то это понимание может принимать очень разные представления. Каждое *представление требований* выполняет определенную задачу, например, служит "мостом", фиксацией соглашения между разными группами специалистов, или используется для оперативного *управления проектом* (отслеживается, в какой фазе реализации находится то или иное требование, кто за него отвечает и пр.), или используется для *верификации* и модельно-ориентированного тестирования. И в первом, и во втором, и в третьем примере мы имеем дело с требованиями, но формализованы они будут *по*-разному.

Итак, формализация требований в проекте может быть очень разной – это зависит от его величины, принятого процесса разработки, используемых инструментальных средств, а также тех задач, которые решают формализованные требования. Более того, может существовать параллельно несколько формализаций, решающих различные задачи. Рассмотрим варианты.

1. *Неформальная постановка требований в переписке по электронной почте*. Хорошо работает в небольших проектах, при вовлеченности заказчика в разработку (например, команда выполняет субподряд). Хорошо также при таком стиле, когда есть взаимопонимание между заказчиком и командой, то есть лишние формальности не требуются. Однако, электронные письма в такой ситуации часто оказываются важными документами – важно уметь вести деловую переписку, подводить итоги, хранить важные письма и пользоваться ими при разногласиях. Важно также вовремя понять, когда такой способ перестает работать и необходимы более формальные подходы.
2. *Требования в виде документа* – описание *предметной области* и ее свойств, *техническое задание* как приложение к контракту, *функциональная спецификация* для разработчиков и т.д.
3. *Требования в виде графа с зависимостями* в одном из средств поддержки требований (*IBM* Rational RequisitePro, *DOORS*, Borland CaliberRM и нек. др.). Такое представление удобно при частом изменении требований, при отслеживании выполнения требований, при организации "привязки" к требованиям задач, людей, тестов, кода. Важно также, чтобы была возможность легко создавать такие графы из текстовых документов, и наоборот, создавать презентационные документы по таким графам.
4. *Формальная модель требований* для *верификации*, модельно-ориентированного тестирования и т.д.

Итак, каждый способ представления требований должен отвечать на следующие вопросы: кто потребитель, *пользователь* этого представления, как именно, с какой целью это *представление* используется.

Некоторые ошибки при документировании требований. Перечислим ряд ошибок, встречающихся при составлении технических заданий и иных документов с требованиями.

• Описание возможных решений вместо требований.

• Нечеткие требования, которые не допускают однозначную проверку, оставляют недосказанности, имеют оттенок советов, обсуждений, рекомендаций: "Возможно, что имеет смысл реализовать также…..", "и т.д.".

• Игнорирование аудитории, для которой предназначено *представление требований*. Например, если спецификацию составляет инженер заказчика, то часто встречается переизбыток информации об оборудовании, с которым должна работать программная система, отсутствует *глоссарий* терминов и определений основных понятий, используются многочисленные синонимы и т.д. Или допущен слишком большой уклон в сторону программирования, что делает данную спецификацию непонятной всем непрограммистам.

• Пропуск важных аспектов, связанных с нефункциональными требованиями, в частности, информации об окружении системы, о сроках готовности других систем, с которыми должна взаимодействовать данная. Последнее случается, например, когда данная программная система является частью более крупного проекта. Типичны проблемы при создании программно-аппаратных систем, когда аппаратура не успевает вовремя и ПО невозможно тестировать, а в сроках и требованиях это не предусмотрено….

Цикл работы с требованиями

В своде знаний *по программной инженерии* SWEBOK определяются следующие виды *деятельности* при работе с требованиями.

• *Выделение требований (requirements elicitation)*, нацеленное на выявление всех возможных источников требований и ограничений на работу системы и *извлечение требований* из этих источников.

• *Анализ требований (requirements analysis)*, целью которого является обнаружение и устранение противоречий и неоднозначностей в требованиях, их уточнение и систематизация.

• *Описание требований (requirements specification)*. В результате этой *деятельности* требования должны быть оформлены в виде структурированного набора документов и моделей, который может *систематически* анализироваться, оцениваться с разных *позиций* и в итоге должен быть утвержден как официальная формулировка требований к системе.

• *Валидация требований (requirements validation)*, которая решает задачу оценки понятности сформулированных требований и их характеристик, необходимых, чтобы разрабатывать ПО на их основе, в первую очередь, непротиворечивости и полноты, а также соответствия корпоративным стандартам на техническую документацию.

Проблема

Например, строители строят дома, пусть разные: многоэтажные, отдельные коттеджи, офисные здания и пр. – однако, весь этот спектр вполне может охватить одна компания. Но все это дома. Строительной компании не приходится строить летающую тарелку, гиперболоид инженера Гарина, *луноход*, систему мгновенной телепортации и пр. А разработчики *ПО*, во многом, находятся именно в таком положении.

Велико разнообразие систем, которые создает одна компания, одна *команда*. Хотя сейчас и намечаются тенденции к специализации рынка разработки *ПО*, однако, причуды мировой экономики и многие другие причины приводят к тому, что строго специализированных компаний не так много, как хотелось бы. Многие области испытывают большой дефицит отдельных программистов и целых коллективов и компаний, хорошо разбирающихся в их специфике. Примером такой области может служить телевидение, где о данной проблеме открыто говорят на заседаниях различных международных сообществ.

Кроме того, *ПО* продолжает проникать во все новые и новые области человеческой *деятельности*, и сформулировать *адекватные* требования в этом случае вообще оказывается супертрудной задачей.

Но даже если речь идет об одной, определенной области, то *процент* новых, уникальных черт систем, принадлежащих этой области, высок: *по сочетанию* пользовательских характеристик, *по* особенностям среды исполнения и требованиям к *интеграции*, *по распределенности информации* о требованиях среди работников компании-заказчика. Все это несет на себе очень большой отпечаток

индивидуальности заказчика – персональной или его компании, – сильно связано со спецификой его бизнеса, используемого в этой области оборудования.

Кроме того, существуют трудности в понимании между заказчиком и программистами, а еще – в изменчивости *ПО* (требования имеют тенденцию меняться в ходе разработки).

В итоге, далеко не очевидно, что та система, которую хочет заказчик, вообще может быть сделана. Трудно найти черную кошку в темной комнате, особенно если ее там нет. Или то, как поняли и воплотили задачу разработчики, окажется удобным, востребованным на рынке.

Ошибки и разночтения, которые возникают при выявлении требований к системе, оказываются одними из самых дорогих. Требования – это то исходное понимание задачи разработчиками, которое является основой всей разработки.

Несколько слов о трудности взаимопонимания заказчика и разработчиков. Здесь сказывается большой разрыв между программистами и другими людьми. Во-первых, потому, что чтобы хорошо разобраться, какой должна быть система *автоматизации* больницы и система поддержки химических экспериментов – надо поработать в соответствующей области достаточное время. Или как-то иным способом научиться видеть проблемы данной *предметной области* изнутри. Во-вторых, сказывается специфичность программирования как сферы *деятельности*. Для большинства пользователей и заказчиков крайне не просто сформулировать точное *знание*, которое необходимо программистам. На вопрос, сколько типов анализов существует в вашей лаборатории, доктор, подумав, отвечает - 43. И уже потом, случайно, программист уточнил, а нет ли других типов? Конечно, есть, ответил доктор, только они случаются редко и могут быть в некотором смысле, какими угодно. В первый же раз он назвал лишь типовые. Но, конечно же, *информационная система* должна хранить информацию обо всех анализах, проведенных в лаборатории….

Теперь чуть подробнее об изменчивости *ПО* и ее причинах.

- Меняется ситуация на рынке, для которого предназначалась система или требования к системе ползут из-за быстро сменяющихся перспектив продажи еще неготовой системы.
- В ходе разработки возникают проблемы и трудности, в силу которых итоговая функциональность меняется (видоизменяется, урезается).
- Заказчик может менять свое собственное видение системы: то ли он лучше понимает, что же ему на самом деле надо, то ли выясняется, что он что-то упустил с самого начала, то ли выясняется, что разработчики его не так поняли. В общем, всякое бывает, важно лишь, что теперь заказчик определенно хочет иного.

Нечего и говорить, что *изменчивость требований по* ходу разработки очень болезненно сказывается на продукте. *Авторы* сталкивались, например, с такой ситуацией, что еще не созданную систему отдел продаж начинает *активно* продавать, в силу чего поступает огромный *поток* дополнительных требований. Все их реализовать в полном объеме не удается, в итоге система оказывается набором демо-функциональности….