Принципы работы с требованиями к программному обеспечению. Унифицированный подход
(главные мысли)

Дин Леффингуэлл, Дон Уидриг

Оглавление

Введение

Глава 1. Проблема требований

Ошибки, обнаруженные на стадии проектирования, могут относится к одной из двух категорий:

1. Ошибки, возникающие при создании технического проекта из правильного множества требований
2. Ошибки, которые следовало обнаружить ранее как ошибки требований, каким-то образом «просочившиеся» на фазу проектирования.

В зависимости от того, где и когда при работе над проектом разработки ПО был обнаружен дефект, цена его может разниться в 50-100 раз.

Заключение:

1. Ошибки в определении требований являются наиболее распространенными
2. Стоимость устранения таких ошибок, как правило, одна из самых высоких

Ошибки в определении требований приводят к затратам, составляющим 25-40% бюджета проекта разработки в целом.

Глава 2. Введение в управление требованиями

Управление требованиями – это систематический подход к выявлению, организации и документированию требований к системе, а также процесс, в ходе которого вырабатывается и обеспечивается соглашение между заказчиком и выполняющей проект группой по поводу меняющихся требований к системе.

Необходимо иметь «карту территории».

Основное – понять потребности заинтересованных лиц.

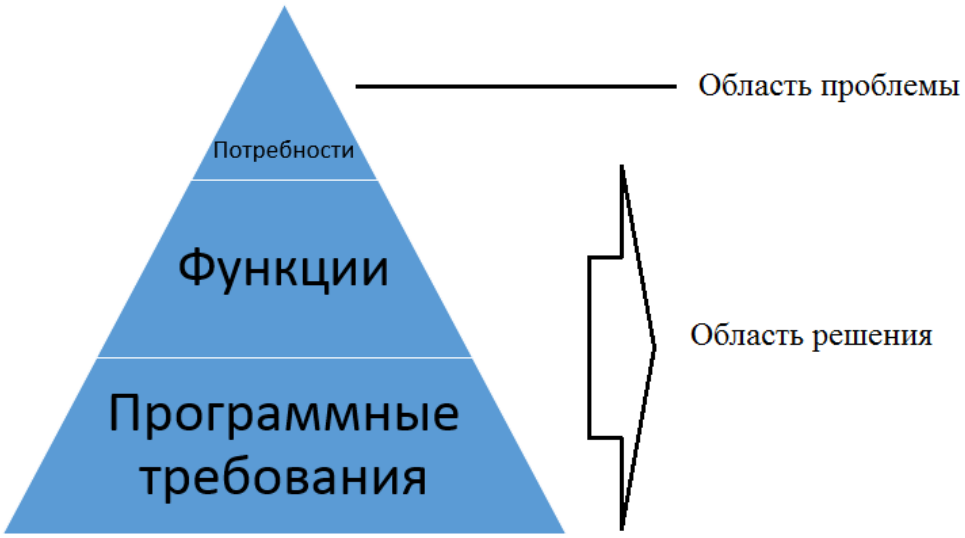
Первым делом полезно сформулировать, что мы узнали в проблемной области и как собираемся этого достичь посредством решения. Создать просто описания на языке клиента, которые мы будем использовать в качестве ярлыков, чтобы пояснить заказчику, как наша система решает его задачу. Это – набор функций.

Функция – это предоставляемое системой обслуживание для удовлетворения одной или нескольких потребностей заинтересованных лиц.

После этого, на основе описанных функций, конкретизируем требования к программному обеспечению.

Прецедент описывает серии взаимодействий пользователь / система, в результате которых пользователь решает некие свои задачи.

Карта территории:



Глава 3. Команда разработчиков

История развития разработки ПО – это история роста ее масштабов.

Процесс управления требованиями, хотя и по-разному, затрагивает каждого члена команды.

Успешное управление требованиями может осуществляться только эффективно организованной командой разработчиков.

Недостаток информации от клиента, неполные требования и спецификации, а также изменения требований и спецификаций наиболее часто указываются в качестве основных причин возникновения проблем в проектах, не достигших поставленных целей.

Часть 1. Анализ проблемы

Команды разработчиков забегают вперед, предлагая технические решения, основанные на неадекватном понимании решаемой проблемы

Глава 4. Пять этапов анализа проблемы

Анализ проблем – это процесс осознания реальных проблем и потребностей пользователей и предложения решений, позволяющих удовлетворить эти потребности.

В некотором смысле проблемы и возможности – это две стороны одной медали; ваша проблема является моей возможностью. Все зависит от точки зрения.

Проблема – это разница между желаемым и воспринимаемым.

Иногда самым простым решением является изменение бизнес-процесса, а не создание новой системы.

Один из путей решения проблем – изменение восприятия у пользователей.

Цель анализа проблемы состоит в том, чтобы добиться лучшего понимания решаемой проблемы до начала разработки.

Этап 1. Достижение соглашения об определении проблемы

Простейший способ – просто записать проблему и выяснить, все ли согласны с такой постановкой (записывать в стандартной форме):

Элемент	**Описание**
Проблема	Описание проблемы
Воздействует на	Указание лиц, на которых оказывает влияние данная проблема
Результатом чего является	Описание воздействия данной проблемы на заинтересованных лиц и бизнес-деятельность
Выигрыш от	Указание предлагаемого решения
Может состоять в следующем	Список основных предоставляемых решением преимуществ

Этап 2. Выделение основных причин – проблем, стоящих за проблемой

Помогает методы анализа корневых причин (диаграммы Исикавы).

Иногда нужно спросить людей, непосредственно занимающихся этим делом, что они считают корневыми причинами. В более сложных случаях – используются другие методы выявления корневых причин.

Устранение корневых причин

Качественные данные свидетельствуют, что многие корневые причины не стоят того, чтобы их устранять, поскольку затраты на их устранение превысят причиняемый проблемам ущерб. Сначала необходимо определить влияние каждой корневой причины. Результат изобразить в виде «Парето-диаграммы».

Пример – в ходе анализа такой Парето-диаграммы было выяснено, что существенно проблемой является неправильные заказы на покупку. Формальная постановка проблемы для нее:

Элемент	**Описание**
Проблема	Неправильных заказов на покупку
Воздействует на	Выполняющий заказы персонал, клиентов, производство, продажи и обслуживание клиентов
Результатом чего является	Увеличение остатков, повышение производственных затрат, неудовлетворенность клиентов, уменьшение прибыли
Выигрыш от	Новой системы, направленной на решение данной проблемы, может состоять в следующем: - Повышение точности заказов на покупку в точке ввода - Совершенствование учета данных о покупках В конечном счете - увеличение прибыли

Данная постановка проблемы согласуется с заказчиками и заинтересованными лицами и доводится до сведения всех разработчиков.

Этап 3. Выявление заинтересованных лиц и пользователей.

Заинтересованные лица – это все, на кого реализация новой системы или приложения может оказать материальное воздействие.

Понимание потребностей пользователей и других заинтересованных лиц является ключевым фактором в выработке успешного решения.

Потребности заинтересованных лиц, не являющихся пользователями, также необходимо выявить и учесть.
Пользователи и лица, заинтересованные в новой системе заказов на покупку:

Пользователи	**Другие заинтересованные лица**
Служащие, занимающиеся вводом заказа	Администратор ИС, команда разработчиков
Руководитель отдела приема заказов	Главный финансист
Контроль производства	Управляющий производством
Служащий, выписывающий счета	

Этап 4. Определение границ системы-решения

Делим мир на две части – наша система и то, что взаимодействует с нашей системой.

Актор – это находящееся вне системы нечто (или некто), взаимодействующее с системой. Изображается в виде пактограммы человечка.

Выявление акторов является ключевым аналитическим этапом в анализе проблемы.

Этап 5. Выявление ограничений, налагаемых на решение

Ограничения уменьшают степень свободы, которой мы располагаем при предложении решения.

Некоторые из обнаруженных ограничений могут стать требованиями к системе.

Все ограничения необходимо записать в таблицу:

Источник	Ограничение	Объяснение

Глава 5. Моделирование бизнес-процессов

При разработке ПО необходимо моделировать бизнес-процессы.

Цель бизнес-моделирования:

- Разобраться в структуре и динамике организации
- Удостовериться в том, что заказчики, конечные пользователи и разработчики имеют одинаковое понимание организации

При правильном выборе метода моделирования бизнес-процесса некоторые рабочие продукты, а именно прецеденты и модели объектов, в дальнейшем будут полезны при создании решения.

Для проектирования ПО и бизнес-процессов используется унифицированный язык моделирования – UML.

Две модельные конструкции для моделирования бизнес-процессов:

— Модель прецедентов бизнес-процесса
— Модель объектов бизнес-процесса.

Преобразование модели БП в модель системы:

— Сотрудники предприятия становятся акторами разрабатываемой системы
— Описанные для сотрудников предприятия варианты поведения можно автоматизировать; это помогает нам выявить прецеденты системы и определить необходимые функциональные возможности
— Сущности бизнес-процесса – это то, в поддержке чего нам должна помочь система, поэтому они помогают нам выявить классы сущностей при анализе модели системы

Глава 6. Инженерия систем, интенсивно использующих программное обеспечение

Системная инженерия – это междисциплинарный подход и соответствующие ему средства, которые позволяют создавать успешно работающие системы (будем использовать для анализа проблем во встраиваемых решениях).

Системная инженерия поможет понять требования, которые будут предъявлены ко всем программным приложениям, выполняемым внутри системы-решения.

Основные принципы системной инженерии:

— Знание проблемы, клиента и потребителя
— Использование основанных на потребностях критериев эффективности для принятия системных решений
— Задание требований и управление ими
— Выявление и оценка альтернатив при выработке решения
— Верификация и проверка правильности требований, а также функционирования решения

— Обеспечение целостности системы
— Использование согласованного и документированного процесса
— Организация действий согласно плану.

Декомпозиция – последовательное разбиение. Показателем того, что дело сделано и сделано «правильно», являются следующие критерии:

— Распределение и разбиение функций оптимизированно так, чтобы добиться достижения общей функциональности системы с минимальными затратами и максимальной гибкостью
— Каждая подсистема может быть определена, спроектирована и построена небольшой или, в крайнем случае, средней командой
— Каждая подсистема может быть изготовлена в рамках физических ограничений и технологий существующих производственных процессов
— Каждую подсистему можно надежно протестировать, причем существует возможность подходящих установок и сопряжений, имитирующих интерфейсы с другими подсистемами
— Соответствующее внимание уделено физической стороне (размеру, весу, размещению и распределению подсистем), которая также оптимизирована исходя из общесистемных соображений.

Нисходящий процесс разработки требований предназначен для того, чтобы гарантировать, что все системные требования выполняются некой определенной подсистемой или совокупностью взаимодействующих подсистем.

Новый тип требований – **производны** требования (производными от процесса декомпозиции):

— Требования к подсистемам
— Требования к интерфейсам

Производные требования не являются требованиями заказчика. Но важно понимать, что спецификация производных требований будет в конечном счете влиять на способность системы выполнять ее работу, а также на удобство эксплуатации и робастность системы.

В настоящее время во многих системах необходимо оптимизировать возможности их программного обеспечения, а не аппаратных компонентов.

Чаще всего происходит следующее – требование подсистем гораздо больше, чем требований конечного потребителя.

При декомпозиции мы так же выделяем акторов для каждой подсистемы (часто одна подсистема будет являться атором для другой). А также описываем ограничения системы.

Часть 2. Понимание потребностей пользователей

Среди причин возникновения проблем в проектах чаще всего упоминается «недостаток информации от пользователя».

Глава 7. Задача выявления требований

Преграды на пути выявления требований:

- Синдром «да, но…»
  - Этот синдром будет всегда
  - Можно заранее выявить все «но»
- Синдром «неоткрытых руин»
  - Чем больше найдено требований, тем лучше известно, что это еще не все
- Синдром «пользователя и разработчика»
  - Пользователи и разработчики, как правило, принадлежат к различным мирам, говорят на разных языках и имеют различный опыт, мотивацию и цели

Смягчить взаимоотношения пользователя и разработчика можно:

Проблема	**Решение**
Пользователи не знают, чего хотят, а если и знают, то не могут это выразить	Признавать пользователя экспертом в предметной области и ценить его в этом качества; пытаться использовать альтернативные методы общения и выявления требований
Пользователи думают, что они знают, чего хотят, до тех пор, пока разработчики не предоставят им то, что они якобы хотели	Как можно раньше предлагать альтернативные методы выявления: раскадровку, ролевые игры, прототипы и тд
Аналитики думают, что они понимают проблемы пользователя лучше его самого	Поставить аналитика на место пользователя. Провести ролевую игру в течении часа или всего дня
Все считают, что другие руководствуются политическими мотивами	Такова человеческая натура, поэтому пусть все остается как есть

Глава 8. Функции продукта или системы

Чтобы добиться успеха, команда разработчиков должна играть гораздо более активную роль в процессе выявления требований.

Совокупность исходных данных, которые мы называем потребностями заинтересованных лиц или пользователей, представляет собой критически важный фрагмент собираемой картины.

Функция – это отражение реальных потребностей.

В функции что («мне нужно») незаметно заменяется на как («что, по моему мнению, должна делать система, чтобы удовлетворить данную потребность»).

Использование функций – удобный способ описания возможностей без лишних подробностей.

Но нужно четко выяснить, какая потребность стоит за функцией.

Функция – обслуживание, предоставляемое системой для выполнения одной или нескольких потребностей заказчика.

От пользователя могут поступать описания потребностей, описания функций. Эта информация может противоречить друг другу. Но разработчики, в конечном итоге, должны выяснить, что система должна делать.

Систему произвольной сложности можно определить с помощью списка из 25 – 99 функций. Но лучше – не более 50.

Функции конкретизируются в требования к программному обеспечению.

После того, как возможные функции перечислены, можно приступить к принятию решений вида «отложить до следующей версии, реализовать немедленно, полностью отвергнуть, исследовать дополнительно». Этот процесс корректировки масштаба лучше проводить на уровне функций, а не на уровне требований, иначе можно просто увязнуть в деталях.

Атрибуты функции – элементы данных, которые обеспечивают дополнительную информацию о каждой функции.

Примеры атрибутов: идентификатор, статус, приоритет (полезность), трудоемкость, риск, стабильность, целевая версия, назначение, обоснование.

Использование атрибутов, в частности, поможет в поиске и приоритезацию функций.

Глава 9. Интервьюирование

Контекстно-свободные вопросы помогают достичь понимания реальной проблемы, не оказывая влияния на ответы пользователя.

Контекстно-свободные вопросы – это вопросы о проблеме пользователя, никак не связанные с возможным решением этой проблемы (Кто является пользователем? Кто является клиентом? Отличаются ли их потребности?).

Ответы на эти вопросы дадут информацию о том, что видит пользователь. На основе его ответов можно предложить одно из готовых решений (которого как бы «выбирает» пользователь). В процессе разговора, когда более-менее понятно, о чем речь – добавляйте вопросы, которые описывают предлагаемое вам решение. На основе ответов получите обратную связь.

В самом конце интервью нужно записать три самых важных проблемы или потребности, которые уяснил тот, кто проводит интервью. После десяти интервью будет около 15 важных проблем – базис будущих требований.

Проводите интервью первым делом! Проводите интервью для каждого нового класса проблем и для каждого нового проекта!

Анкетирование – это не интервьюирование.

Анкетирование можно использовать для проверки предположений и сбора статистических данных о предпочтениях.

Глава 10. Совещания, посвященные требованиям

Преимущества совещания, посвященного требованиям:

- Оно помогает создать команду, подчиненную одной общей цели – успеху данного проекта
- Все заинтересованные лица получают возможность высказать свое мнение, никто не остается в стороне
- Оно формулирует соглашение между заинтересованными лицами и командой разработчиков по поводу того, что должно делать приложение
- Оно может высветить и разрешить политические вопросы, которые влияют на успех проекта
- Результат, предварительное определение системы на уровне функций, немедленно становится известным.

Надлежащая подготовка является залогом успеха совещания.

Если постарайтесь, найдите ведущего, не являющегося членом команды. Если ведущий – член команды, он *не должен* вносить свои предложения, чтобы не нарушить правильную атмосферу совещания.

Существует несколько проблем совещания: опоздания, трудность возобновления работы после перерыва, доминирование отдельных участников, недостаток предложений от участников, негативные комментарии, мелочное поведение, ослабление энергии после перерывов и обеда.

Для их решения можно использовать специальные карточки (5 минут на речь, Отличная идея, Бесплатный выстрел и тд).

Глава 11. Мозговой штурм и отбор идей

Мозговой штурм состоит из двух фаз: генерация идей и их отбор.

Ведущий обязательно должен обозначить цель мозгового штурма. Когда (по цели) уже никто не высказывается – собрание нужно заканчивать.

Главное правило мозгового штурма – идеи критиковать нельзя!

Каждый член команды записывает свои идеи своими словами на своем листочке (чтобы не потерять, чтобы запись шла быстрее, и чтобы идея была написана именно так, как хочет автор идеи).

После этапа генерации идей начинается этап отбора идей. Он состоит из:

- — отсечения (убирание бредовых идей),
    - Бредовые идеи – это индикатор свободного мышления. Чем больше таких идей – тем более свободны были люди при генерации идей.
- — группировки (одинаковые идеи, идеи из смежных областей),
- — определения функций (автор идеи описывает ее одним предложением – чтобы не было различных трактовок),
- — расстановки приоритетов:
    - Накопительное голосование (срабатывает только один раз, повторно использовать в одном и том же проекте не получится)
    - Разбиение на категории – критическая (обязательная), важная (программой не захотят пользоваться), полезная (было бы хорошо ее иметь) – каждый участник должен задать каждой идее одну

категорию. В итоге критические умножаются на 9, важные – на 3, полезные – на 1. Баллы суммируются.

Глава 12. Раскадровка

Раскадровка нужна для раннего обнаружения синдрома «да, но …» и «неоткрытых руин».

Три типа раскадровок:

- Пассивные – пользователь сам всё рассказывает. Используются экранные копии, бизнес-правила, выходные отчеты
- Активные – разработчики монтируют фильм из потенциальных функциях системы и показывают пользователям
- Интерактивные – демонстрация макетов пользователю. Впоследствии, эти макеты (куски кода) могут быть выброшены из проекта

Раскадровки для ориентированных на пользователя систем описывают три основных элемента любой деятельности:

1. Кто такие игроки?
2. Что с ними происходит?
3. Как это происходит?

Раскадровки могут быть настолько разнообразны, насколько позволяет воображение членов команды.

Рекомендации при создании раскадровки:

- Не вкладывайте много средств в раскадровку (лучше сделать раскадровку не реально, а схематичной)
- Если вы ничего не меняете, вы ничему не научитесь
- Не делайте раскадровку слишком хорошей
- Если возможно, делайте раскадровку интерактивной

Рекомендации для разработчиков:

- Создавайте раскадровку как можно раньше
- Делайте раскадровку чаще
- Применяйте раскадровки во всех проектах, имеющих новое или новаторское содержание

Глава 13. Применение прецедентов

Прецедент описывает последовательность действий, выполняемых системой с целью предоставить полезный результат конкретному актору.

Прецеденты пишутся на естественном языке пользователя. Их легко описывать и документировать. Так же можно задать концептуальное описание интерфейса системы.

Прецеденты не очень подходят для определения нефункциональных аспектов системных требований.

Пример. Прецедент: перераспределение товаров на складе

1. Заведующий дает команду перераспределить предметы на складе
2. Заведующему предлагается окно на рис XXX
3. Товары можно упорядочить различными способами, которые представлены в меню:
    a. Упорядочить по алфавиту
    b. Упорядочить по значению индекса
    c. Упорядочить по срокам хранения
4. Таблица «Откуда» позволяет выбрать режим рассмотрения (можно рассматривать все помещения склада или только те помещения, где есть указанный товар)

Прецеденты рассказывают о том, как система взаимодействует с пользователем при предоставлении своих функциональных возможностей.

Глава 14. Обыгрывание ролей

Обыгрывание ролей – на час или на пол дня выполнять работу пользователей. Но, во многих сложных случаях не просто обыгрывать роль пользователя (пилот Боинга, оператор АЭС и тд). В таких случаях применяется сценарный просмотр.

Сценарный просмотр – это исполнение роли на бумаге. Как в театре. В таком случае сразу видно, когда исполнителям ролей не хватает информации.

CRC-карточки – Класс-Обязанность-Взаимодействие). Каждому участнику выдается набор индексных карточек, описывающих класс (объект); обязанности (поведение); а также взаимодействия (с какими из моделируемых сущностей взаимодействует объект).

Когда актер-инициатор инициирует определенное поведение, все участники следуют поведению, заданному на их карточках. Если процесс прерывается из-за недостатка информации или если одной сущности необходимо

переговорить с другой, а взаимодействие не определено, то карточки модифицируются, и роли разыгрываются снова.

Можно использовать метод CRC-карточек и для программирования ООП-систем.

Глава 15. Создание прототипов

Прототипы помогают в преодолении синдромов «да, но…» и «неоткрытых руин».

Виды прототипов:

- Отбрасываемые – нужен для понимания осуществимости проекта. По окончании прототип выбрасывается;
- Эволюционирующие – прототип реализуется в той же архитектуре, что и рабочий проект, так что можно прототип развивать дальше;
- Операционные
- Вертикальные – создает довольно мало требований, но делает это качественно;
- Горизонтальные – создаем широкий спектр функциональных возможностей системы;
- Пользовательские интерфейсы – создание интерфейса пользователя, а не бизнес-логики приложения;
- Алгоритмические

Прототип требований к ПО – это частичная реализация системы ПО, созданная с целью помочь разработчикам, пользователям и клиентам лучше понять требования к системе.

Хорошо известные и понимаемые требования не нуждаются в прототипировании, если они не являются необходимыми для того, чтобы помочь пользователям визуализировать содержание других их потребностей.

Неизвестные потребности являются теми «неоткрытыми руинами», которые мы хотели бы найти, но их прототипировать неизвестно. Поэтому прототипировать будем «неясные» требования – требования по которым плохо определены.

Пользователи должны работать с прототипом в той среде, в которой они будут работать с реальным ПО – так откроются факторы среды.

Прототип должен быть как можно более дешевым.

Часть 3. Определение системы

Объем информации, которой мы должны управлять, увеличивается по мере спуска к основанию пирамиды потребности в ПО.

Глава 16. Организация информации о требованиях

Требования необходимо записывать и документировать.

Спецификация требований к системе (или приложению) описывает ее внешнее поведение. С ней сверяются и на нее ссылаются все участники проекта. Но не обязательно это будет один документ, чаще говорят о пакете требований к ПО.

Процесс проектирования системы сам по себе создает новые классы требований – из-за разбиения системы на подсистемы.

Сначала разрабатываются требования к системе в целом. Затем – происходит разбиение системы на подсистемы и разрабатываются спецификации требований для каждой из подсистем. Эти спецификации должны полностью описывать внешнее поведение подсистем (без учета их разбиения на подсистемы следующего уровня). Так появляются производные требования – требования не системы, а конкретной подсистемы.

Лучше всего записывать все требования в документ (и текущие, и требования на будущее), но необходимо четко выделять те их них, которые планируется реализовать в текущей версии.

MRD – документ требований маркетинга – чтобы упростить общение между руководством, службой маркетинга и разработчиками.

Глава 17. Документ-концепция

Документ-концепция описывает приложение в общих чертах, а также содержит описания целевых рынков, пользователей системы и функций приложения.

Обязательными элементами документа-концепции являются следующие:

— Общая информация и введение
— Сведения о пользователях системы и описание обслуживаемых рынков, функций, которые предполагается реализовать в версии 1.0
— Прочие требования (регуляторные и требования среды)
— Будущие функции, которые были выявлены, но не вошли в версию 1.0

При реализации документа-концепции версии 2.0 лучше использовать Delta Vision – документ изменений концепции. В нем отражается то, что изменилось, а также любая другая информация, которую следует включить для ясности. Основное внимание в нем уделяется тому, что нового включено в данную версию и что отличает ее от предыдущих версий.

Крайне редко практикуется документирование полных требований крупномасштабной существующей системы.

Глава 18. Лидер продукта

Хотя не существует общих предписаний, которым можно следовать при выборе лидера проекта, для каждой команды чрезвычайно важно найти его, поддержать или наделить полномочиями того, кто уже фактически стал лидером. После этого задача команды состоит во всесторонней помощи лидеру в управлении требованиями к приложению. Это поможет добиться успеха.

Лидер может создать совет по контролю за изменениями. Именно этот совет может добавлять или удалять функции в версии по инициативе лидера.

Часть 4. Управление масштабом

Прежде чем расширять команду, разрабатывать более подробные спецификации, окончательно формулировать технологические идеи проектирования и создавать сценарии тестирования, следует остановиться и научиться управлять масштабом проекта.

Глава 19. Проблема масштаба проекта

Масштаб проекта:

— Набор функций
— Ресурсы
— Время на реализацию

Закон Брукса (1975) – дополнительное привлечение рабочей силы к запаздывающему программному проекту приведет к еще большему его запаздыванию.

Если необходимый для реализации требуемых функций объем работ равен имеющимся ресурсам, умноженным на выделенное время, масштаб проекта является достижимым и проблем не возникает.

Так как часто руководство оценивает проект в 200% трудозатрат (то есть, за отведенный период времени и ресурсов можно сделать только половину проект), команда проекта должна оценивать масштаб, как до его начала, так и во время проведения разработки. Однако в типичном случае задача состоит в сокращении масштаба: Если мы действительно начнем разработку с 200-процентным масштабом, необходимо сократить масштаб проекта, как минимум, в два раза, чтобы иметь шанс на успех.

Глава 20. Задание масштаба проекта

Базовый уровень – это разбитое на элементы множество функций или требований, которые намечено реализовать в конкретной версии приложения.

Базовый уровень должен обладать следующими свойствами:

— Должен быть приемлемым, как минимум, для заказчика
— Должен иметь разумную вероятность успеха с точки зрения команды разработчиков.

Первый шаг – перечислить функции, которые были определены для приложения (их не должно быть слишком мало или слишком много) и расстановках их в порядке приоритета.

Шаг второй – определение приблизительного уровня трудозатрат для каждой из предлагаемых функций (определяем примерный порядок величины). Нужно стараться тратить не очень много времени на этот этап.

Шаг третий – оценка риска, связанного с каждой функцией. Риск дает возможность оценить, какими могут быть последствия включения конкретной функции в базовый уровень проекта.

Наша задача в том, чтобы применить ограниченные ресурсы на наибольшей выгодой для клиента. Именно по этой расставляем функции по приоритету и выбираем их в соответствии с ним.

Как следует из опыта, базовый уровень можно ограничить критическими функциями, одной-двумя важными и, на усмотрение команде, включение полезных функций. Ненаучно, зато работает. То, что осталось «за бортом» базового уровня – функции на будущее.

Глава 21. Умение общаться с заказчиком

Мы можем активно привлекать наших заказчиков к управлению их требованиями и масштабом их проекта, чтобы обеспечить как качество, так и своевременность разработки ПО.

Если сокращение масштаба произошло без заказчика – его нужно проинформировать об этом.

Руководящий принцип при управлении масштабом: «меньше обещать и больше делать».

Большее становится врагом достаточного. Лучшее – враг хорошего!

Удовлетворительное выполнение задания: в установленное время предоставить достаточно функциональных возможностей для удовлетворения реальной потребности клиента.

Базовый уровень функций обеспечивает удобный механизм управления высокоуровневыми изменениями. Рассмотрим официальное изменение, когда заказчик запрашивает новую возможность системы, не являющуюся частью базового уровня. Если команда проекта изначально тщательно определила базовый уровень, то следует исходить из предположения, что любое изменение в базовом уровне повлияет на ресурсы, график или набор функций, которые должны быть представлены в данной версии.

Если ресурсы фиксированы и график изменить нельзя, команда проекта должна привлечь заказчика к процессу принятия решения о приоритете новой функции по отношению к другим функциям, определенным для реализации в данной версии.

Официальные изменения заказчика – одна из самых легких проблем управления требованиями. Существуют еще и неофициальные изменения, они обсуждаются в главе 34.

Глава 22. Управление масштабом и модели процесса разработки ПО

Процесс разработки ПО определяет, кто (какой член команды), что (какие действия), когда (данные действия по отношения к другим действиям) и как (детали и этапы этих действий) делает для достижения цели.

«Водопадная модель»

Предпосылка появления: устранение тенденции переходить непосредственно к кодированию, не имея адекватного представления о требованиях к системе.

В ней не указывалось необходимости создания прототипа, как и было принято делать.

Так же отводилась большая роль разработке требований. Эта модель стала олицетворять застывший косный подход к разработке, когда требования «заморожены» на время жизни проекта, изменения запрещены, а процесс разработки «живет» своей собственной жизнью.

Управление масштабом при такой модели затруднено. В результате – опять началась тенденция программировать без требований.

«Спиральная модель»

Для тех, кто верит, что путь к успеху состоит из инкрементных разработок на основе рисков.

Сначала создаются серии основанных на рисках прототипов, а затем проводится структурированный процесс построения конечной системы (аналогично модели водопада).

Проекты с 200% масштабом так же сложно реализовать в спиральной модели, но, к сроку сдачи – будет готов один или два прототипа и будет известна реакция заказчика.

«Итеративный подход»

Состоит из четырех фаз:

- Начало – понимание бизнес-варианта проекта, определяется масштаб, производится анализ проблемы, создается документ-концепция, оценивается график, бюджет и факторы риска;
- Исследование – уточняются требования к системе, задается исходная выполняемая архитектура, разрабатывается и демонстрируется ранний прототип;
- Построение – реализация проекта, завершается проектирование и доработка архитектуры;
- Внедрение – бета-тестирование, протестированная базовая версия передается сообществу пользователей и разворачивается для использования.

Проект разрабатывается итеративным инкрементным методом. Ранние итерации следует разрабатывать для оценки жизнеспособности выбранной

архитектуры, для некоторых самых важных прецедентов и прецедентов с высоким приоритетом.

Для каждой итерации задаются план и критерии оценки. На итерацию команда тратит столько времени, сколько нужно. То есть, итерация – это мини-водопад. В результате выполнения итерации получается полноценная версия программы.

Преимущества:

1. Раннее получение «да, но…» - при каждой следующей итерации размеры «да, но…» уменьшаются;
2. Легче управлять масштабом – если в первых итерациях процент невыполнения составляет от 30% и выше – масштаб проекта завышен, нужно сокращать. Даже при срыве срока, в конце у заказчика будет несколько готовых программ, а последнюю – даже можно будет развернуть и показывать клиентам.

Независимо от того, какая модель используется, команда должна предложить по крайней мере один устойчивый оценочный прототип для раннего получения реакции клиента.

Раннее получение реакции «да, но…» является одной из самых главных задач в процессе разработки ПО.

Часть 5. Уточнение определения системы

Собранные требования (и только они!) служат движущей силой проекта. Если окажется, что они недостаточны или даже ошибочны, их нужно быстро и официально изменить.

Глава 23. Требования к ПО

Мы переходим к разработке функций системы до уровня детализации, достаточного, чтобы гарантировать, что в ходе проектирования и кодирования будет создана система, полностью соответствующая потребностям пользователя. При этом мы переходим на следующий уровень конкретизации и создаем более полную, глубокую модель требований к разрабатываемой системе. Кол-во информации, которой надо управлять, тоже увеличивается.

Согласно Дэвису, для полного определения системы, необходимо описать пять основных категорий элементов:

1. Вводы системы – необходимо не только указать содержимое ввода, но и, если нужно, подробно описать устройства, а также протокол ввода;
2. Выводы системы – описать поддерживаемые устройства вывода, протокол и формы генерируемой системой информации;
3. Функции системы – отображение вводов и выводов и их различные комбинации;
4. Атрибуты системы – надежность, удобство сопровождения, доступность и пропускная способность, которые должны учитывать разработчики;
5. Атрибуты системной среды – способность системы работать в условиях операционных ограничений и нагрузок, а также совместимость с ОС.

Полный набор требований к ПО можно задать, определив эти пять элементов. Мы сможем оценить, является ли некая «вещь» требованием к ПО, проверив, соответствует ли она данному подробному определению.

Чтобы предоставить пользователю некую функцию, разработчики должны выполнить одно или несколько конкретизированных программных требований.

Функции – это абстрактность для упрощения обсуждения системы на высоком уровне. Программный код на их основе не написать.

Требования не должны содержать информацию, к требованиям не относящуюся: подробности проектирования, необязательные ограничения реализации, способы тестирования, информация об управлении проектом и т.д.

Так же нужно игнорировать требования к ПО (например – реализация на VB.NET), если это – прихоть разработчика. Если этого хочет заказчик, то, скорее всего, это требование.

В идеале: сначала нужно производить разработку требований, а затем (зная, что делать) – производить проектирование.

В реальности деятельности по разработке требований и проектированию должны осуществляться итеративно. Выявление требований, их определение и принятие проектных решений циклически чередуются. Процесс заключается в непрерывном круговороте:

> Имеющиеся требования приводят к выбору определенных вариантов проектирования, а те, в свою очередь, могут инициировать новые требования.

В данном случае (это суть итеративного метода) необходимо правильно понимать потребности пользователя и привлекать его к процессу разработки требований. Именно в этом случае получится проводить эффективно управлять требованиями в итеративном подходе.

Типы требований:

- Программные требования
  - Функциональные требований к ПО
  - Нефункциональные требования к ПО
- Ограничения проектирования

Функциональные требования – ориентированы на действия. Когда пользователь делает Х, система будет делать У.

Нефункциональные требования (нужны для описания атрибутов системного окружения):

- практичность (субъективное понятие, зависит от точки зрения наблюдателя. Пример формализации –билль о правах пользователей, Карат, 1998);
- надежность (доступность, среднее время между отказами, среднее время восстановления, точность, максимальный коэффициент ошибок, количество различных ошибок – незначительные, серьезные, критические);
- производительность (время ответа для транзакции, пропускная способность, емкость, режимы снижения производительности);
- возможность обслуживания (способность легко модифицировать ПО с целью внесения изменений и исправлений).

Ограничения проектирования (Налагаются на проект системы или процессы, с помощью которых система создается. Они не влияют на внешнее поведение системы, но должны выполняться для удовлетворения технических, деловых или контрактных обязательство):

- Некое требование, которое допускает несколько вариантов проектирования – проект является осознанным выбором среди этих вариантов;
- Требование, налагаемое на процесс создания программы;
- Различные корпоративные, отраслевые, государственные и международные стандарты.

Золотая середина учета ограничений проектирования:

- Следует отличать их от других требований;

- Лучше включить все ограничения проектирования в специальный раздел пакета требований или использовать спец.атрибут;
- Необходимо указывать источник каждого ограничения проектирования;
- Следует документировать объяснения для каждого ограничения проектирования.

Предпочтительнее двигаться вперед (пусть и с несовершенной организацией), чем топтаться на месте, разрабатывая план совершенного разбиения требований по категориям.

Дочерние требования:

Они служат для повышения уровня конкретизации, выраженного в родительском требовании. Пользователи смотрят спецификацию верхнего уровня (чтобы понять требования), а разработчики уточняют их в подробных дочерних спецификациях. Можно расширить иерархию до такого уровня детализации, в котором нуждается продукт. Но слишком много уровней делать так же не стоит.

Глава 24. Уточнение прецедентов

Методологию прецедентов нужно использовать, если:

- Система является функционально ориентированной; существует несколько различных типов пользователей и функционального поведения;
- Команда реализует систему, используя UML и объектно-ориентированные методы.

Нежелательно использовать модель прецедентов в следующих системах:

- Где пользователей мало или нет вообще, а интерфейсы минимальны;
- В которых доминируют нефункциональные требования и ограничения проектирования.

Прецедент – это описание множества действий (включая варианты), которые система выполняет для того, чтобы доставить полезный осязаемый результат определенному актору.

Методология прецедентов выделяет два элемента, которые должны присутствовать во всех реализациях прецендентов:

1. Прецедент;

2. Актор – это некто или нечто, взаимодействующее с системой (не являются частью описываемой системы, а находятся за ее пределами). Бывает три типа акторов:
    a. Пользователи
    b. Приборы
    c. Другие системы

Детализация прецедентов не является декомпозицией системы. Разработка прецедентов похожа на уточнение последовательностей действий, а не на иерархическое разбиение их на более мелкие действия.

Прецедент может состоять из нескольких действий актора с системой (например – загрузка бутылок в машину и получение квитанции – это один прецедент, а не два, так как по отдельности эти действия для актора бессмысленны).

Имя прецедента должно быть уникальным, состоять из глагола, показывающего, что делает система.

Сердцевина прецедента – поток событий. Как правило, это текстовое описание операций актора и различных ответов системы. Поток событий описывает, что, как предполагается, будет делать система в зависимости от поведения актора. Главное – обеспечить понимание, и не существует единственного подхода на все случаи жизни.

Поток событий не указывает, как система выполняет действия. Он указывает только, что происходит.

Бывают и альтернативные потоки событий (например – закончилась бумага, и напечатать квитанцию невозможно). В первую очередь нужно описать основной поток событий, затем – альтернативные события. Обязательно нужно прописать условия, приводящие к ним.

Существуют пред и постусловия прецедентов. Предусловия – это те события, которые предшествуют наступлению прецедента. Постусловия – то состояние системы, которое должно быть сохранено по завершению прецедента (даже если он шел по альтернативному пути).

Глава 25. Спецификация к программному обеспечению (Modern Software Requirements Specification).

Выявленные и детализированные требования к ПО, документы, прецеденты и модели – совокупность данных артефактов представляет полную концептуальную модель создаваемой системы.

На данном этапе мы подошли к созданию концептуальной модели системы (или посредника – proxy – системы).

SRS – спецификация программных требований. Современный Пакет Спецификаций Требований к ПО – Modern SRS Package.

Modern SRS Package – это пакет информации, полностью описывающий внешнее поведение системы, т.е. набор артефактов следующего содержания: «Вот то, что система должна делать, чтобы предоставить эти функции»:

- Он служит основой для общения между всеми сторонами-участниками
- Формально или неформально, он представляет соглашение между этими различными сторонами
- Служит в качестве справочника стандартов для администратора ПО
- Чтобы принять решение о том, что должен делать код – разработчики должны ознакомиться с этим документом
- Нужен для групп выполняющие тестирование и гарантирующих качество (QA) – пакет служит в качестве стандарта при планировании и выполнении тестов
- Он управляет развитием системы на фазе построения.

Не важно, кто пишет пакет. Гораздо важнее, чтобы он существовал и являлся основой для предстоящих действий по построению и тестированию.

Как организовать пакет? Главное, чтобы команда в нем сама разбиралась. Примерная структура пакета:

```
Общая информация
История пересмотров
Содержание
1. Введение
   1.1.    Цель
   1.2.    Масштаб
   1.3.    Ссылки
   1.4.    Предположения и зависимости
2. Краткая характеристика модели прецедентов
3. Краткая характеристика акторов
4. Требования
   4.1.    Функциональные требования
   4.2.    Нефункциональные требования
      4.2.1.    Практичность
      4.2.2.    Надежность
      4.2.3.    Производительность
      4.2.4.    Возможность сопровождения
5. Требования к интерактивной пользовательской документации и
   системе подсказок
```

```
6. Ограничения проектирования
7. Закупаемы компоненты
8. Интерфейсы
   8.1.   Пользовательские интерфейсы
   8.2.   Аппаратные интерфейсы
   8.3.   Программные интерфейсы
   8.4.   Коммуникационные интерфейсы
9. Требования лицензирования
10.  Примечания об авторских правах и т.д.
11.  Применяемые стандарты
Индекс
Глоссарий
Приложение. Спецификация прецедентов
     История пересмотров прецедента
```

Дата	Версия	Описание	Автор

```
     Название прецедента
     Краткое описание
     Потоки событий
          Основной поток
          Альтернативный поток
          Первый альтернативный поток
          Второй альтернативный поток
     Специальные требования
          Первое специальное требование
          Второе специальное требование
     Предусловия
          Предусловие 1
     Постусловия
          Постусловие 1
     Точки расширения
          Название точки расширения
     Другие материалы прецедента
```

Глава 26. Неоднозначность и уровень конкретизации

На вопрос: «Какой уровень конкретизации необходимо обеспечить?», можно ответить следующим образом: «Это зависит от содержания приложения и того, насколько те, кто выполняет реализацию, способны принять правильные решения или хотя бы задать вопросы там, где есть неоднозначность».

Важно найти «золотую середину», это зависит от умения команды.

Пример неоднозначности – «Мери имела маленького барашка».

Один из способов уменьшить неоднозначность – применять метод «формальной» спецификации требований.

Рекомендации к нахождению «золотой середины»:

— Используйте везде, где это возможно, естественный язык
— Используйте рисунки и диаграммы, чтобы лучше проиллюстрировать сущность
— Если сомневаетесь – спрашивайте! Даже если сомнений нет, все равно старайтесь спрашивать
— В тех случаях, когда неверное понимание недопустимо, дополните ваши спецификации более формальными методами.

Глава 27. Критерии качества требований к ПО

Высококачественный пакет Modern SRS Package должен быть:

— Корректным;
— Недвусмысленным;
— Полным;
— Непротиворечивым;
— Упорядоченным по важности и стабильности;
— Поддающимся проверке;
— Модифицируемым;
— Трассируемым;
— Понимаемым.

Требования должны быть верифицируемыми (тестируемыми), то есть, каждое требование в процессе тестирования должно быть проверены – соответствующим образом должны быть подготовлены тесты.

Множество требований является модифицируемым тогда и только тогда, когда его структура и стиль таковы, что любое изменение требований можно произвести просто, полно и согласовано, не нарушая существующей структуры и стиля всего множества.

Трассируюмые требования – имеют номер или идентификатор (к ним можно обратиться в процессе разработки). То есть, в любой момент можно просмотреть как субтребования, которые вытекают из верхнего уровня; а так же можно ознакомиться с верхним требованием, зная какое-то субтребование. Например, требования 63.1, 63.2, 63.3 – все происходят от требования 63.

Трассировка решает вопрос «что, если?» - что, если мы сейчас изменим это требование? Как оно отразится и на каких требованиях?

Критерии качества пакета Modern SRS Package:

— Хорошо составленное оглавление
— Хороший индекс
— История исправлений
    - Номер и код исправления
    - Дата исправления
    - Краткое описание
    - Имя человека, ответственного за исправление
    - Добавить маркер исправления к требования (есть/нет)
— Глоссарий (термины, аббревиатуры, мнемокоды и т.д.)

Глава 28. Теоритически обоснованные формальные методы спецификации требований

Можно использовать следующие:

— **Псевдокод** – основан на смеси естественного и алгоритмического (программерского) языка;
— **Конечные автоматы** – систему можно рассмотреть как гипотетическую машину, которая в конкретный момент времени может находиться только в одном из указанных состояний. Поэтому можно и выводы, и следующее состояние определить, основываясь на знании текущего состояния и события, вызывающего транзакцию;
— **Деревья и таблицы решений** – используется при комбинации вводов (когда много входных условий, типа: если входа А истина и, при условии что вход В тоже истина, а вход Д – лож, тогда вывод Х истина, а вывод У – ложь). Либо в таблице или с помощью диаграммы это отображаем;
— **Диаграммы деятельности** (блок-схемы) – алгоритмические способы представления (например, UML); понятно для пользователей, но сложно обновлять руками – не хочется перерисовывать;
— **Модели сущность-**связь – если набор требований описывает структуру данных; непонятно для пользователя, но является мощным инструментом моделирования;
— **Объектно-ориентированные модели** – если нужно описать взаимодействие сущностей; язык UML;
— **Схемы потоков данных** – DFD-диаграммы; сложны для понимания (но не так, как ER-модель), мощное средство моделирования.

По возможности, в пакете требований желательно не использовать формальный подход к описанию требований, либо – как можно меньше. Так же желательно использовать как можно меньше различных формальных методов в одном пакете. В идеале – использовать только один.

Когда дело касается сопровождения, на первом план должен быть здравый смысл. Желательно обновлять весь пакет после изменения. Если это невозможно – ставить пометку «устаревший» на некритических требованиях. Критически важные требования обязательно нужно обновлять после изменения.

Нужно использовать автоматизированные средства ведения пакета требования.

Часть 6. Построение правильной системы

В главе есть ответ на вопрос: «Как определить, что мы создаем «правильную систему»?

Глава 29. Как правильно построить «правильную» систему: общие положения

Общие положения:

- Необходимо постоянно убеждаться, что разработка находится на правильном пути;
- Необходимо убеждаться в корректности результатов работы;
- Необходимо научиться обрабатывать изменения, возникающие в процессе разработки.

Верификация – постоянно выполняемый процесс проверки того, что каждый шаг разработки является корректным (есть различные толкования данного термина).

Верификация (по IEEE 1012-1986) – процесс оценивания системы или компонента с целью определить, удовлетворяют ли результаты некой фазы условиям, наложенным в начале данной фазы.

Затраты на верификацию могут быть очень большими. Нужно оценить, какие части системы верифицировать обязательно, а какие – не слишком «дорого» и верифицировать их. Но верификацией придется заниматься на каждой стадии разработки. Она необходима чтобы удостовериться в правильности следующих переходов:

- От потребностей пользователей к функциям продукта;
- От функций продукта к требованиям;
- От требований к архитектуре;

— От архитектуры к модели проектирования;
— От модели проектирования к реализации;
— От реализации к планированию тестов.

Верификация очень важна на стадии проектирования, так как возникшие на этой стадии ошибки очень сложно устранять на стадии реализации.

Работа по проверке правильности разработки (валидация) имеет два важных аспекта:

1. Показать, что продукт соответствует предъявляемым к нему требованиям
2. Сосредоточить внимание на приемке клиентом конечного результата

Глава 30. От понимания требований к реализации системы

Проблема ортогональности – отсутствие прямой связи между отражающими проблемную область требованиями и реализационным кодом.

Цель моделирования состоит в том, чтобы упростить систему до понимаемой «сути».

Программная архитектура представляет собой описание элементов, из которых конструируется система, взаимодействий между ними, а также образцов, согласно которым они составляются, и ограничений, налагаемых на эти образцы.

С помощью архитектуры решаются следующие задачи:

— Понять, что делает система
— Понять, как она работает
— Иметь возможность обдумывать и разрабатывать части системы
— Расширять систему
— Повторно использовать часть (части) системы при создании новых систем.

Архитектурные модели нужны различным группам заинтересованных лиц, которые будут рассматривать предложенную архитектуру с разных точек зрения.

Различным участникам нужны различные представления системы (строительство дома – модель дома для монтажников, кровельщиков, электриков, водопроводчиков и т.д.).

Представление системной архитектуры:

1. Логическое представление
2. Вид с точки зрения реализации
3. Вид с точки зрения процессов
4. Вид с точки зрения развертывания

Язык UML содержит специальные модельные конструкции, которые поддерживают реализацию прецедентов. Прецедент можно разделить на структурную часть (классы, элементы, интерфейсы и подсистемы) и поведенческую (динамика взаимодействия элементов для получения результата). Если все это описать – перейти к реализации будет намного проще.

Глава 31. Использование трассировки для поддержки верификации

Рекомендуется проводить верификацию постоянно.

— Трассировка – степень, до которой можно установить связь между двумя или большим числом продуктов процесса разработки, особенно продуктами, которые являются по отношению друг к другу главным-подчиненным, например, степень соответствия между требованиями и проектом конкретного программного компонента.

— Трассировка – степень, до которой каждый элемент продукта программной разработки оправдывает смысл своего существования; например, насколько каждый элемент схемы, изображенной кружками и стрелками, соответствует удовлетворяемому им требованию.

Существует явная и неявная трассировка. Явная – команда указывает на связь элементов. Неявная – например, иерархическая связь (указывать такую не нужно).

Трассировка зачастую помогает понять другие части проекта. То есть, ее можно использовать, чтобы понять отношения между элементами проекта.

Всегда следите за балансом важности элементов трассировки и затрат на их поддержку.

Матрица отношений трассировки:

— Если у некой строки не удалось обнаружить никаких отношений трассировки – тут есть потенциальная ошибка;

— Если у столбца нет зависимого отношения – это ошибка, нужно исправлять (возможно не правильно поняли код, требования и т.д.).

Для реализации трассировки можно использовать электронные таблицы и БД. Но для крупных проектом обязательно нужно использовать специализированные автоматизированные средства.

При верификации важно убедиться, что ни одна связь не пропущена и все элементы более низкого уровня, такие как программные требования / прецеденты, надлежащим образом связаны с более высокоуровневыми требования к продукту.

Связи, как и сами требования, будут эволюционировать со временем.

Верификация требует вдумчивого отношения и нельзя полагаться на чисто механические действия.

Глава 32. Проверка правильности системы

Валидация – процесс оценивания системы или компонента во время или по окончании процесса разработки с целью определить, удовлетворяет ли она указанным требованиям.

То есть, проверка правильности призвана подтвердить, что реализованная система соответствует заданным требованиям.

Приемо-сдаточные испытания привлекают заказчика к окончательной проверке системы.

Высокого качества можно добиться, только протестировав систему на соответствие ее требованиям.

Трассировка помогает убедиться, что тестовые примеры покрывают все требуемые функции.

Желательно использовать матрицы трассировки тестовых примеров.

Многие требования ограничений проектирования необходимо тестировать как можно проще. Например, требование – «Писать на VB.NET» проверяется просмотром исходного кода.

Проверка правильности позволяет убедиться, что система работает так, как предполагалось, т.е. удовлетворяет как документированным требованиям клиента, так и реальным сценариям использования.

Валидация производится постоянно.

Глава 33. Применение метода анализа дивидендов для определения объема V&V-действий

Планировать V&V-действия будем, исходя из ответов на следующие два вопроса:

1. Какими будут социальные или экономические последствия сбоя системы?
2. Какой необходим объем V&V-действий, чтобы гарантировать, что эти последствия не возникнут?

Глубина определяет уровень детализации при проведении проверки элемента системы.

Методы проверки:

— Просмотр;
— Сквозной контроль – структурированный просмотр, осуществляющийся широкой группой лиц;
— Независимые ревизии – то же, что и предыдущие методы, но просмотр осуществляют независимые специалисты;
— Тестирование черного ящика;
— Прозрачное тестирование – требует значительный объем ресурсов.

Покрытие определяет, какая часть элементов системы подвергается верификации и проверке правильности.

Варианты проведения V&V:

— Вариант 1. Верифицировать и проверять правильность всех элементов.

  Допустимо выборочное проведение V&V, если известно, в чем состоят риски. Если элементы не подвергаются V&V, важно документировать причины этого.

— Вариант 2. Анализ рисков для определения необходимости V&V

  Анализ рисков служит руководством при выборе элементов проекта для проведения V&V.

Всегда необходимо выполнять V&V для аспектов, имеющих отношение к безопасности людей.

Глава 34. Управление изменениями

Внешние факторы (контролировать невозможно):

— Произошли изменения проблемы;
— Пользователи изменили свое мнение;
— Изменилась внешняя среда;
— Вошла в строй новая система (сам факт ввода в строй новой системы выявляет новые требования к ней).

Процесс управления требованиями может быть полезен только в том случае, если он позволяет выявлять и решать проблему изменений. Невозможно предотвратить изменения, но можно научиться ими управлять.

Внутренние факторы:

— Не всем нужным людям задали вопросы (или задавали не те вопросы) при первоначальном выявлении требований;
— Не удалось создать процесс управления требованиями («заморозка» требований, либо все могли изменить всё, что захотят).

Требования, полученные из неофициальных источников, могут составлять до половины масштаба всего проекта!

Просочившиеся требования – те, которые вошли в систему незаметно для членов команды, отвечающих за график, бюджет и критерии качества.

Процесс управления требованиями дисциплинирует команду. Процесс обработки изменений должен включать в себя следующие шаги:

1. Осознать, что изменения неизбежны, и разработать план управления изменениями;
2. Сформировать базовый уровень требований;
3. Установить единый канал контроля изменений – изменение системы нельзя инициировать до тех пор, пока механизм контроля за изменениями не признает его «официальным»;
4. Использовать систему контроля изменений для их фиксации – команде следует разработать некую систему для фиксации всех запросов на изменение;
5. Обработать изменения по иерархическому принципу.

Упорядочение процесса внесения изменений не означает, что можно уносить огромное количество всевозможных изменений.

Пожелания заказчиков о внесении изменений не предполагают официального изменений графика и бюджета.

Нам придется принять сознательное решение о том, какие недостатки оставить в системе.

Программист не имеет права от имени пользователя вводить новые функции и требования непосредственно в код.

Каждая новая функций/требование оказывает влияние на стоимость, график, надежность и риск, связанные с проектом.

Процесс рассмотрения и принятия изменений в некоторых организациях носит название «контроль изменений», «контроль версий», или управление конфигурацией.

Желательно вести контрольный журнал изменений – вести его в хронологической последовательности, записывать все изменения и их атрибуты. В идеале – вести такой журнал автоматически.

Глава 35. С чего начать

Не существует единственно верного способа осуществления управления требованиями.

Рецепт

Шаг 1. Понимание решаемой проблемы

1. Выполните состоящий из пяти этапов процесс анализа проблемы
   1.1. Достигнуть соглашения по определению проблемы;
   1.2. Выделить основные причины – проблемы, стоящие за проблемами;
   1.3. Выявить заинтересованных лиц и пользователей;
   1.4. Определить границу системы решения;
   1.5. Выявить ограничения, которые необходимо наложить на решение
2. Доведите до сведения внешних участников постановку проблемы и убедитесь, что вам удалось достигнуть согласия по определению проблемы, прежде чем двигаться дальше

Шаг 2. Понимание потребностей пользователей

1. Создайте структурированное интервью, используя в качестве образца шаблон из части 2, модифицированный применительно к конкретному приложению;
2. Проведите интервью с 5-15 пользователями/заинтересованными лицами, которые были выявлены на шаге 1;
3. Подведите итоги интервью, сформулировав 10-15 наиболее часто упоминавшихся потребностей или используя «метод выразительных цитат»; т.е. запишите 10-15 особенно запомнившихся высказываний участников, в которых их потребности писаны их собственными словами;
4. Используйте цитаты или сформулированные вами описания потребностей для начала создания пирамиды требований. Начинайте задание трассировки требований;
5. Проведите совещание по вопросу требований к проекту. Используйте как общие, так и связанные с конкретным проектом подготовительные документы;
   5.1. Проведите сеанс мозгового штурма, чтобы выявить/уточнить функции
   5.2. Выполните отсечение идей и определите приоритеты функций
   5.3. Используйте классификацию функций, чтобы определить их как критические, важные и полезные
6. Проводите совещание один или два раза в год, чтобы обеспечить постоянный приток структурированных мнений заказчика;
7. Создайте раскадровки для всех инновационных концепций. Представьте их и покажите соответствующее множество прецедентов пользователям, чтобы удостовериться, что вы правильно их поняли;
8. Старайтесь сделать так, чтобы ваш процесс представлял пользователю хотя бы один оценочный прототип, который пользователи могут тестировать в своей среде.

Шаг 3. Определение системы

1. Воспользуйтесь понятием документа-концепции и создайте его шаблон применительно к нуждам вашего проекта;
2. Сформулируйте определение позиции продукта. Ознакомьте с ним всех участников и убедитесь, что они с ним согласны. Если это не так, остановитесь и добейтесь согласия. Обязательно убедитесь в согласии ваших клиентов;
3. Введите все выявленные на шаге 2 и полученные от разработчиков и представителей маркетинга функции в документ-концепцию. Произведите их обратную трассировку к потребностям пользователей. Используйте в документе-концепции атрибуты приоритета (критический, важный, полезный), риска (высокий, низкий, средний), трудоемкости (командо-

месяцы), стабильности (высокая, низкая, средняя) и реализации (v 1.0 и т.д.);
4. Разработайте иллюстративные прецеденты в приложении документа-концепции, чтобы его функции были всем понятны;
5. Сделайте документ-концепцию «живым» документом проекта. Опубликуйте его, чтобы было легче его использовать и пересматривать. Автоматически сделайте ответственного за документ-концепцию официальным каналом изменения функций. Используйте документ Delta Vision. Следует сделать так, чтобы ваша компания имел соответствующий современному состоянию проекта документ-концепцию.

Шаг 4. Постоянное управление масштабом и контроль изменений

1. На основе выполненных командой оценок трудозатрат определите базовый уровень для каждой версии документа-концепции, используя атрибут «номер версии»;
2. Следует достигать соглашения с заказчиком относительно масштаба. Помогите команде принять жесткие решения по масштабу и двигайтесь дальше;
3. Придерживайтесь принципов итеративной разработки и обучайте этим принципам команду. Повсюду обсуждайте и корректируйте ожидания;
4. Управляйте изменениями, используя базовый уровень. Для фиксации новых функций, возникающих в результате нормального течения событий, используйте документ Delta Vision. Убедитесь, что все предложенные функции записаны и ничего не потеряно. Уполномочьте совет по контролю за изменениями принимать жесткие решения;
5. Установите систему управления запросами изменений, чтобы фиксировать все запросы на изменения и гарантировать, что они поступают через эту систему в совет по контролю за изменениями.

Шаг 5. Уточните определения системы

1. Обеспечьте наличие на всех этапах спецификации требований к ПО (используйте организационную форму пакета Modern SRS Package), которая задает полный набор функционального и нефункционального поведения продукта. Разработайте подробные прецеденты для основных функций системы;
2. Пусть этой задачей займется группа разработчиков или группа тестирования. Помогите им получить необходимые навыки и помогайте по мере необходимости. Используйте формальные методы анализа только в тех случаях, когда неправильное понимание недопустимо;
3. Осуществите трассировку требований к прецедентам и функциям и обратно;

4. Убедитесь, что заданы все нефункциональные требования к системе. Используемы образец поможет удостовериться, что вы задали необходимые вопросы.

Шаг 6. Построение правильной системы

1. Проведите анализ и оценку рисков, чтобы определить, для каких элементов неправильная реализация недопустима. Разработайте план действий по верификации и проверке правильности, основываясь на результатах этих оценок;
2. На этом этапе привлекайте к решению задач управления требованиями отдел тестирования. Пусть сотрудники отдела подключаются к планированию тестов с самого начала. Группа тестирования должна разработать тестовые процедуры и примеры, которые трассируются к прецедентам, а также функциональным и нефункциональным требованиям;
3. Если в вашей компании есть независимый отдел гарантии качества, ему должна отводиться роль в отслеживании и оценке процесса управления требованиями и плана V&V;
4. При создании модели проектирования следует полагаться на прецеденты и их реализацию, чтобы связать элементы проекта с требованиями;
5. Обеспечьте периодическое проведение приемочных испытаний по окончании значительных этапов, чтобы проверить правильность результатов работы и обеспечить постоянное участие заказчиков.

Шаг 7. Управление процессом работы с требованиями

1. Лидер должен нести персональную ответственность за документ-концепцию, еженедельно (вместе с командой) оценивать состояние дел и регулярно готовить отчеты и запросы, способствующие этим действиям;
2. Отлеживайте процесс спецификации требований к ПО, чтобы убедиться, что концепция надлежащим образом осуществляется в детализированных требованиям;
3. Осуществляйте руководство процессом контроля за изменениями, который проводит ССВ, чтобы гарантировать, что перед тем, как разрешить внесение существенных изменений в систему, производится оценка поступивших предложений.

Шаг 8. Примите наши поздравления! Вы выпустили продукт!