УДК 004.05 DOI: 10.31799/2077-5687-2024-5-52-59

НЕПРЕРЫВНАЯ ДОСТАВКА И ИНТЕГРАЦИЯ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

А. А. Олимпиев
Санкт-Петербургский государственный университет аэрокосмического приборостроения

В статье рассмотрены проблемы управления требованиями, имеющие место в системах разработки программного обеспечения. Проведен анализ причин возникновения ситуаций, когда реализация требования не соответствует пожеланиям. Выделены причины роста технического долга. В качестве решения проблемы предложено создание автоматизированной системы непрерывной доставки и интеграции требований, которая позволит более эффективно управлять данными процессами.

Ключевые слова: управление требованиями, технический долг, жизненный цикл программного обеспечения, управление качеством программных изделий.

Для цитирования:
Олимпиев, А. А. Непрерывная доставка и интеграция требований к программному обеспечению / А. А. Олимпиев // Системный анализ и логистика. – 2024. – № 5(43). – с. 52-59. DOI: 10.31799/2077-5687-2024-5-52-59.

CONTINUOUS DELIVERY AND INTEGRATION THE SOFTWARE REQUIREMENTS

A. A. Olimpiev
St. Petersburg State University of Aerospace Instrumentation

The article discusses the problems of requirements management that take place in software development systems. The analysis of the causes of situations when the implementation of the requirement does not meet the wishes is carried out. The reasons for the growth of technical debt are formulated. As a solution to the problem, it is proposed to create an automated system for continuous delivery and integration of requirements, which will allow more efficient management of these processes.

Keywords: requirements management, technical debt, software lifecycle, quality management of software products.

For citation:
Olimpiev, A. A. Continuous delivery and integration the software requirements / A. A. Olimpiev // System analysis and logistics. – 2024. – № 5(43). – p. 52-59. DOI: 10.31799/2077-5687-2024-5-52-59.

Введение

В практике разработки программного обеспечения достаточно часто возникают проблемы формулировки и последующей доставки требований до разработчиков, а также интеграция новых требований в бизнес-логику. Основными из этих проблем можно свести к следующей группе типовых ситуации:

- требований и задач, поступающих разработку;
- не соответствие сформулированных требований и задач реальному пожеланию заказчика;
- отсутствие единого языка описания требований и, как следствие, неоднозначная их интерпретация;
- чересчур абстрактная формулировка требования и задачи;
- противоречие между требованиями разных заказчиков или требованиями, поступившими ранее;
- необоснованный отказ в реализации требования и задачи.

Проблемы формулировки, анализа и обработки требований изучаются специалистами давно (см. например, [1]), поэтому перечисленные ситуации являются известными и имеют под собой причины, которые возникают на различных этапах процесса формулировки и

доставки требований до разработчика. Поиск способов устранения проблем осуществляется достаточно интенсивно и за последние годы было найдено много различных решений, вышла четвертая версия международного стандарта SWEBOOK (Software Engineering Body of Knowledge), однако в большинстве организаций освоение данных подходов идет крайне медленно и проблемы остаются нерешенными. Для понимания процесса и поиска способов смягчения проблем, а также постепенного перехода на современные технологии, необходимо рассмотреть каждый из этапов формулировки и доставки требования по отдельности.

1. Содержание процесса формулировки, доставки и интеграции требований

1.1. Сбор информации и предварительный анализ требования к программному обеспечению

В соответствии опытом лучших практик и рекомендациями ведущих разработчиков этот этап должен выполняться в тесном взаимодействии с заказчиком [2, 3], который рассказывает о существующей потребности в автоматизации или оптимизации технологического, или бизнес-процесса, или операции.

Задачей бизнес-аналитика, который ведет диалог с заказчиком, является максимально точно сформулировать суть потребности. Даже если заказчик достаточно хорошо понимает, что именно ему нужно, и обладает базовыми знаниями в разработке программного обеспечения, он, скорее всего, не знает особенности конкретной автоматизированной системы и все детали ее реализации.

Современные заказчики обладают следующими особенностями:

- у заказчиков есть доступ через ресурсы Интернет к потенциально безграничному объему информации, которую они постепенно с достаточно большой скоростью осваивают;
- заказчики обладают большим опытом использования автоматизированных систем массового применения, характеризующихся очень высоким качеством (таких как высоко технологичные текстовые редакторы, умные устройства, современные средства вычислительной техники, операционные системы и другие);
- заказчики для организации своей повседневной деятельности и решения рабочих задач используют специальное программное обеспечение, без которого не обходится ни одно современное предприятие, поэтому внедрение программного обеспечения для работы не является чем-то новым, но может заменять существующее;
- заказчик осведомлен, что существует большое многообразие аналогов прикладного программного обеспечения, которые потенциально можно использовать для замены тех инструментов, которые у него есть в наличии, поэтому имеет место конкуренция между разработчиками.

Эти обстоятельства приводят к необходимости соглашаться на проработку любого требования, которое поступает от заказчика, в том числе давать обещания на их выполнение в разумные сроки.

Ввиду объективных причин у бизнес-аналитика и у заказчика отсутствует достаточно времени на формулировку совместного решения сразу, поэтому каждое требование формулируется в общем виде с некоторыми дополнительными пояснениями, а формулировка самого предложения по решению откладывается и передается исполнителям следующих этапов процесса доставки до разработчиков.

Источником ошибок сбора требований является низкая осведомленность участников диалога во всех связях между бизнес-процессами, дефицит времени на сбор информации, отрыв от текущего состояния бэклога продукта, пропуск несущественных моментов или типовых вопросов, недостаточное внимание к потребности заказчика.

Причинами сложившейся ситуации являются большие объемы информации, которые непрерывно поступают от различных заказчиков, большое разнообразие бизнес-процессов,

которые подвергаются автоматизации, а также множественные пересечения этих бизнес-процессов. Ситуация усугубляется тем, что одни и те же бизнес-процессы и операции у различных заказчиков могут выполняться по-разному, что не всегда берется в расчет, а производится попытка сформулировать одно универсальное решение для всех.

На этом этапе также может возникать ситуация, когда одно и то же требование или уточнение к требованию попадают к разным бизнес-аналитикам, которые могут быть не осведомлены о результатах работы друг друга.

1.2 Обработка и определение ценности требования, формулировка User story

Как уже было отмечено выше в процессах анализа требований и разработки всегда имеет место дефицит времени. По этой причине каждое требование, поступившее от заказчика, записывается как есть, после чего через некоторое время анализируется и уточняется. В редких случаях User story может быть сформулировано сразу и отправлено менеджеру проектов для того, чтобы он определил сроки и выделил ресурсы на реализацию. Эта ситуация возможна только в том случае, если требование имеет очень высокий приоритет и суть его очевидна: например, заказчик обнаружил ошибку в программе и не может выполнять свою деятельность без ее устранения. Оптимизация всех процессов управления требованиями обычно выполняться для того, чтобы темп поступления и реализации требований мог опережать или быть соизмеримым с темпом возникновения и удовлетворения потребностей. Но как увеличение темпа разработки без использования соответствующих инструментов приводит к большему искажению информации.

По этим и другим причинам как правило поступившее от заказчика требование обрабатывается повторно: изучаются записи диалога, происходим анализ сценариев работы пользователя и сбор информации о требованиях, имеющих близкое содержание, обсуждение данного требования с коллегами, проводимое в форме сбора мнений, опросов или мозгового штурма.

В результате происходит уточнение, обобщение и согласование у заказчика формулировки требования. Представление его в виде User story и UseCase модели [4], к которому приложены пояснения, обоснования, данные аналитики и описание ценности требования — все необходимое для того, чтобы принять решение о сроках реализации и важности требования.

Основными источниками ошибок на данном этапе являются:

- форсирование отдельных этапов анализа (например, бизнес-аналитик не обсудил детали с коллегами, просмотрел не все записи);
- разрыв во времени между сбором и обработкой информации (прошло много времени после работы с заказчиком и «несущественные» детали были забыты);
- обработку первичных данных делает не тот человек, который собирал информацию;
- недостаток собранной ранее информации.

1.3 Оценка важности требования, определение сроков и трудоемкости реализации

После того, как задача прошла обработку аналитиками, она попадает в бэклог продукта и ждет своего часа, когда руководитель проекта ее обработает и направит в разработку. Поскольку руководителей проекта значительно меньше, чем других сотрудников, то при высокой интенсивности взаимодействия с заказчиком, в бэклоге продукта образуется очередь требующих планирования задач.

Выбирая очередную задачу из бэклога руководитель проекта должен быстро понять суть требования и определить место его в системе. После этого осуществляется оценка сложности и трудоемкости решения задачи, ее важность и связь с другими задачами. На основании разбора этих параметров руководитель проекта принимает решение о срочности и задачи, а также о количестве ресурсов, которые нужно выделить для ее решения [5].

Из-за многообразия требований, которые содержатся в бэклоге продукта и объема поясняющей информации, собранной аналитиками, на проработку каждой задачи у руководителя проекта уходит много времени. Чтобы сократить сроки доставки задачи до разработки, руководитель проекта может может начать торопить аналитиков и требовать уменьшения количества информации, которая должна поступать в бэклог продукта, что приводит к дополнительным ошибкам. Это часто встречающиеся последствия применения Agile, в манифесте которого утверждается, что «Люди и взаимодействие важнее процессов и инструментов», а также «Работающее программное обеспечение важнее исчерпывающей документации» [6].

На этом этапе большие задачи дополнительно могут разбиваться на части и перерабатываться с учетом комплексного их рассмотрения.

Основными источниками дополнительных ошибок являются:

- осознание возникшей ответственности за ценность результата;
- психологическое давление;
- большой объем рутинной работы;
- локальность содержания каждого требования, не учитывающая особенности бизнес-процессов в целом и связи между ними;
- применение приемов упрощения требований и реализации минимально жизнеспособного продукта;
- вынужденное переключение между разными задачами и предметными областями.

1.4. Постановка задачи на разработку

Когда начинается новая итерация разработки, наиболее важные задачи из бэклога продукта переносятся в очередь разработки. Каждая задача, которая находится в очереди разработки, вначале просматривается системным аналитиком, который осуществляет наиболее сложную работу — перевести язык пользователя на язык программистов. В результате сценарий работы пользователя должен быть превращен в системное решение [7] и множество алгоритмов работы программного обеспечения, которые будут реализованы программистами.

Основные трудности, с которыми сталкивается системный аналитик следующие:

- поскольку системный аналитик является техническим специалистом, он, как правило, плохо разбирается в предметной области, с которой он работает;
- системный аналитик смотрит на требование со стороны архитектуры и текущего состояния программного обеспечения, поэтому реализация может оказаться значительно более сложной, чем считает руководитель проекта и бизнес-аналитик;
- задачи, которые с точки зрения разработки должны делаться раньше, имеют более низкий приоритет с точки зрения бизнеса и могут поступать на много позднее необходимого;
- из-за большого разрыва во времени между бизнес-анализом и системным анализом конкретного требования человек, который изначально над ней работал, может забыть ее суть или изменить свое мнение по некоторым вопросам;
- недостаток информации в задаче и неоднозначные формулировки;
- наличие подразумеваемых требований, которые не включили в описание.

Эти и многие другие особенности процесса проектирования создают огромную пропасть между разработчиками и пользователями и являются источниками многих ошибок.

1.5. Реализация требования в программном обеспечении

В конечном итоге программисту приходит постановка задачи на разработку, которая представляет из себя клубок противоречий, заключенный в длинном описании ненужных деталей. Вторая крайность, с которой сталкивается программист — чересчур лаконичное

описание того, что нужно сделать, в котором отсутствует описание причины, почему нужно сделать так, а не иначе. Это приводит к конфликтам, либо снижению мотивации программиста и, следовательно, неправильной реализации.

Очевидно, что для того, чтобы программист мог реализовать именно то, что нужно заказчику, должны выполняться следующие условия:

- требование программисту понятно;
- задача и ее решение с достаточной точностью расписаны;
- в задаче приведена ценность и важность требования;
- в задаче описан результат, который должен быть получен;
- программист не придумывает ничего сверх того, что нужно сделать, и не искажает полученную информацию.

Однако практика показывает, что идеальная ситуация, при которой пользователь получает именно то, что заказал, случается крайне редко.

2. Суть проблемы доставки требований и пути ее решения

2.1. Прослеживаемость требований

Разрыв в представлениях о работе системы и разработке у пользователя и программиста является давно известной проблемой, которую разными способами пытаются решить специалисты уже многие годы.

В частности, современных стандартах одним из важнейших показателей системы менеджмента качества разработки программного обеспечения, который позволяет сократить расстояние между пользователем и программистом, является прослеживаемость требований [8]. Этот показатель можно также охарактеризовать как общая обоснованность принимаемых решений. Прослеживаемость можно определить, как долю всех отработанных требований, по которым можно проверить обоснованность принятых решений, к общему количеству отработанных требований.

Другой способ определить прослеживаемость — рассчитать долю требований, в которых модели предметной области и программного обеспечения не противоречат друг другу и это можно проверить. На данном способе управления качеством основана методология IDEF и раскрыт в методе IDEF6 [9].

Очевидно, что чем выше показатель прослеживаемости требований, тем эффективнее работает организация и тем выше удовлетворенность пользователей, для которых разработка программного обеспечения становится более понятной, а разработчики — заслуживающими доверия.

Именно поэтому улучшению данного показателя и владельцами предприятий, и руководителями проектов уделяется много внимания.

2.2. Проблема и существующие способы ее решения

Основной причиной снижения прослеживаемости требований являются ошибки, которые возникают в работе людей. Эти ошибки описаны выше и возникают из-за ряда типовых причин:

- недостаток информации о реальной потребности пользователя;
- большой разрыв во времени между сбором информации и формулировкой задачи;
- применение неправильных приемов управления людьми и нагнетание обстановки;
- форсирование этапов анализа и наличие подразумевающихся требований;
- терминологический и семантический разрыв в знаниях бизнес-аналитиков и разработчиков.

В большинстве общедоступных источников знаний приводится описание, как должно быть, чтобы эти проблемы не возникали, но практически нет информации о том, как достичь

нужного эффекта, либо информация сильно рассредоточена и ее трудно собрать воедино. Также в большей мере уделяется внимание реализации программного продукта, его тестированию и внедрению, а не организации процессов сбора, доставки и интеграции требований.

Основными рекомендациями, которые должны помочь создать необходимые условия сводятся к следующим моментам:

- наличие регламента на разработку требования и дисциплинированность в его применении;
- применение обучения и user story review;
- увеличение количества встреч, во время которых происходит обсуждение задач;
- кросс-функциональные команды, в которых каждый программист одновременно является тестировщиком, архитектором, системным аналитиком, бизнес-аналитиком и техническим писателем;
- применение таких практик экстремального программирования, как программирование в паре и вовлечение заказчика в процесс разработки;
- написание подробной документации.

При этом подразумевается, что каждый из участников описанного процесса:

- обладает развитым навыком обработки неполной и недостоверной информации;
- является очень эрудированным в разных предметных областях;
- имеет большой опыт разработки именно этой системы;
- ошибается крайне редко, не устает и никогда не жалеет себя;
- быстро осваивается в новом;
- является человеком добросовестным, обладающим высокой внутренней мотивацией;
- и является «супергероем».

В практике для того, чтобы команда была успешной, достаточно 2-3 человека из 10, которые обладают высокой квалификацией, однако практически невозможно добиться, чтобы при росте проекта не накапливался технический долг, который превращает систему в плохо управляемую, что приводит к и усугубляется текучкой кадров.

2.3. Автоматизация процесса доставки и интеграции требований как способ решения проблемы

Предлагаемым способом решения описанной выше проблемы может являться создание автоматизированной системы непрерывной доставки и интеграции требований. Основными особенностями данной системы должны быть:

- наличие интерактивной документации на систему и предметную область;
- возможность автоматической обработки типовых требований;
- возможность анализа распределения потоков требований между компонентами;
- быстрый поиск подобных требований и решений;
- наличие автоматических подсказок для формирования требований;
- интеграция с системами искусственного интеллекта;
- интеллектуальная система поиска противоречий;
- интеграция системы доставки требований с системой разработки;
- визуализация показателей качества разработки на интерактивной архитектуре.

Последнее требование особенно важно для принятия решений относительно приоритетов рефакторинга компонентов и выработки стратегий сокращения технического долга. Концептуальная модель системы непрерывной доставки и интеграции требований показана на рисунке 1.

На рисунке изображен процесс перемещения требований, поступающих от заказчика в разработку. На начальном этапе требования собираются из разных источников и дополняются необходимой для обработки информацией, связями с другими требованиями и сформулированными задачами. Если в системе реализованы алгоритмы автоматизированной формализации требований, то процедура формирования связей может сводиться к простому автоматическому поиску информации в базе знаний, что существенно упростит процесс первичной обработки.

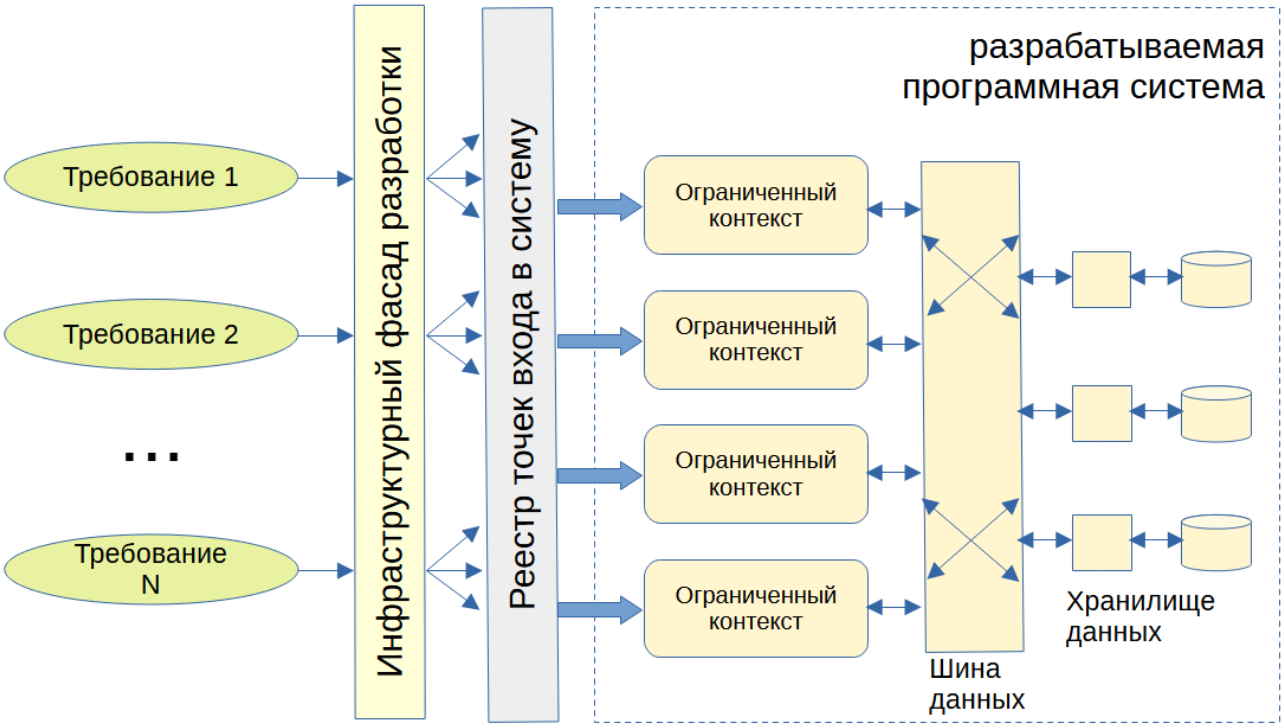

Рис. 1. Концептуальная модель автоматизированной системы

На следующем этапе требование преобразуется в задачу и разбивается на подзадачи, каждая из которых автоматически может быть связана с программным компонентом, в котором она должна быть реализована.

И, наконец, задача передается команде разработке, отвечающей за разработку конкретного компонента, после чего могут формироваться отчет о нагрузке этой команды и рекомендации о необходимости рефакторинга компонента.

Заключение

К преимуществам внедрения автоматизированной системы доставки и интеграции требований являются:

- снижение количества дублей в задачах;
- уменьшение противоречий в документации и задачах;
- прослеживаемость требований;
- ускорение процесса доставки требований;
- управляемый рефакторинг;
- возможность управления техническим долгом и т.п.

Очевидно, что мгновенно создать и внедрить подобную систему невозможно, однако постепенное движение в этом направлении позволит заметно улучшать качество разрабатываемого программного продукта и будет способствовать росту удовлетворенности потребителей.

СПИСОК ЛИТЕРАТУРЫ

1. *Буч Г.* Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон; пер. с анг. Мухин М. –2-е изд. – Москва: ДМК Пресс, 2006. – 496 с.
2. *Орлов С. А.* Программная инженерия: учебник для вузов. / С. А. Орлов. – 5-е изд. – СПб.: Питер, 2016. – 640 с.
3. *Вигерс К.* Разработка требований к программному обеспечению / К. Вигерс, Д. Битти; пер. с англ. – 3-е изд., дополненное – Москва: Издательство «Русская редакция», Санкт-Петербург: БВХ-Петербург, 2014. – 736 с.
4. *Ларман К.* Применение UML 2.0 и шаблонов проектирования */ К. Ларман;* пер. с анг. – 3-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 736 с.
5. *Мартин Р.* Чистый Agile. Основы гибкости / Р. Мартин – СПб.: Питер, 2020. – 352 с.
6. *Коул Р.* Блистательный Agile. Гибкое управление проектами с помощью Agile, Scrum и Kanban / Р. Коул, Э. Скотчер – СПб.: Питер, 2019. – 304 с.
7. *Косяков А.* Системная инженерия. Принципы и практика / А. Косяков, У. Свит; пер. с англ. под ред. В. К. Батоврина. – М.: ДМК Пресс, 2017. – 624 с.
8. ГОСТ Р ИСО 9000-2015. Системы менеджмента качества. Основные положения и словарь. – Москва, ФГУП «СТАНДАРТИНФОРМ», 2015. – 53 с.
9. *Цуканова О. А.* Методология и инструментарий моделирования бизнес-процессов: учеб. пособие / О. А. Цуканова – СПб.: Университет ИТМО, 2015. – 100 с.

ИНФОРМАЦИЯ ОБ АВТОРЕ

Олимпиев Алексей Александрович
канд. техн. наук
Санкт-Петербургский государственный университет аэрокосмического приборостроения
Россия, 190000, Санкт-Петербург, ул. Большая Морская, д.67, лит. А
E-mail: olimpiev@guap.ru

INFORMATION ABOUT THE AUTHOR

Olimpiev Aleksey Aleksandrovich
PhD. tech. Sciences
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaia str., Saint-Petersburg, 190000, Russia
E-mail: olimpiev@guap.ru