С уважением, IBM

ограниченная серия от IBM

Системная инженерия для "чайников"™

Вы узнаете,

- что такое системная инженерия
- как при помощи системной инженерии можно разработать интеллектуальные интегрируемые продукты
- как уменьшить время выхода на рынок, обеспечить быстрое реагирование бизнеса на требования рынка и разрабатывать высококачественные интеллектуальные продукты
- как сократить расходы

Кэтлин Шамие

INCOSE (Международный совет по системной инженерии) выражает искреннюю радость в связи с появлением в серии *Для «чайников»* литературы, которая свидетельствует о растущем интересе к системной инженерии в современном обществе. Совет стремится к распространению знаний об этой области, и нам особенно приятно, что первое издание этой книги совпало с 21-ым международным симпозиумом INCOSE, на котором она была представлена мировому сообществу специалистов по системной инженерии. Мы надеемся, что эта книга станет первой из тех многих книг, руководств и журналов по этим и другим темам, что знакомят читателей с системной инженерией.

Системная инженерия для "чайников"™

ОГРАНИЧЕННАЯ СЕРИЯ ОТ IBM

Кэтлин Шамие

При содействии:

Барклая Брауна
руководителя отдела глобальных решений - IBM Software, Rational

Грега Гормана
руководителя программы решений на базе системной инженерии - IBM Software, Rational

Ханса-Питера Хофмана, доктора философии,
главного методолога по созданию систем - IBM Software, Rational

Брайана Т. Нолана, доктора философии,
руководителя отдела маркетинга, аэрокосмические исследования и средства обороны
IBM Software, Rational

Стива Шоафа
руководителя отдела маркетинга, системная инженерия
IBM Software, Rational

WILEY

Системная инженерия для «чайников»,® ограниченная серия от IBM

Опубликовано
John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030-5774
www.wiley.com

Опубликовано John Wiley & Sons, Inc., Indianapolis, Indiana

Чтобы получить информацию о других наших продуктах и услугах, обращайтесь в отдел Business Development Department в США по телефону +1 317-572-3205. За подробностями процесса создания новых книг серии Для «чайников» для вашей организации обращайтесь по адресу info@dummies.biz. Сведения о лицензии на бренд *For Dummies* в отношении продуктов и услуг можно получить по адресу BrandedRights&Licenses@Wiley.com.

ISBN: 978-1-118-83277-6 (pbk); 978-1-118-83344-5 (ebk)

Произведено в США

10 9 8 7 6 5 4 3 2 1

WILEY

Благодарности

Издательство благодарит всех, кто участвовал в создании этой книги. За подробностями процесса создания новых книг серии Для «чайников» для вашей организации обращайтесь по адресу info@dummies.biz. Сведения о лицензии на бренд *For Dummies* в отношении продуктов и услуг можно получить по адресу BrandedRights&Licenses@Wiley.com.

Мы хотели бы отдельно поблагодарить тех людей, без помощи которых эта книга бы не появилась:

отдел закупок, редакторский отдел и отдел развития средств массовой информации

Редактор проекта: Керри А. Берчфилд

Управляющий редактор: Рев Менгле

Представитель отдела коммерческого развития: Сью Блессинг

Специалист по публикации новых проектов серии: Майкл Салливан Набор

Старший координатор проекта: Кристи Риз

Набор

Старший координатор проекта: Кристи Риз

«Чайники» технологического отдела

Ричард Суодли, вице-президент и ведущий издатель

Энди Каммингс, вице-президент и издатель

Мэри Беднарек, исполнительный директор, руководитель отдела закупок

Мэри К. Кордер, шеф-редактор

«Чайники» отдела подписки

Кэтлин Небенхаус, вице-президент и ведущий издатель

Набор

Дебби Стэйли, руководитель

Валидация

Анатолий Волохов, валидатор русского перевода

Отдел коммерческого развития

Лиза Коулман, руководитель отдела освоения новых рынков и развития брендов

Содержание

Введение

Интеллектуальные устройства — неотъемлемая часть нашей жизни. С их помощью мы отслеживаем почтовые отправления, управляем светофорами, самолетами и находим нужный маршрут. Это ключевой элемент систем и услуг, которыми мы пользуемся каждый день, — от "умных" телефонов и автомобилей, до технологий, применяемых в медицине, аэрокосмической и оборонной промышленности.

Интеллектуальные, технически оснащенные и взаимосвязанные устройства выводят нас на принципиально новый уровень взаимодействия друг с другом и окружающей средой. Различные комбинации элементов электроники, датчиков, программного и аппаратного обеспечения помогают нам создавать все новые многофункциональные устройства, учитывающие потребности каждого из нас. А безграничные фантазия и изобретательность позволяют нам разрабатывать сотни инновационных, специализированных и персонифицированных систем и услуг.

При разработке интеллектуальных устройств главная задача — это правильная организация: мы должны максимально эффективно интегрировать сложные комбинациии различных технологий, чтобы реализовать интеллектуальную и комплексную "систему систем", которая будет соответствовать поставленным целям. Все это стновится возможно благодаря такому понятию, как системная инженерия.

Данная книга *Системная инженерия для «чайников»* (ограниченная серия от IBM) дает понимание системной инженерии и того, как она может помочь вам разрабатывать интегрированные интеллектуальные продукты. Если вы стремитесь быстро завоевать рынок, оперативно реагировать на изменяющиеся тенденции, создавать интеллектуальные продукты высокого качества с минимальными затратами, ограниченная серия от IBM *Системная инженерия для «чайников»* — это именно то, что вам нужно.

Структура книги

Книга состоит из семи глав, которые помогут вам понять проблемы, на решение которых направлена системная инженерия, а также изучить все этапы их решения. Ознакомьтесь с кратким содержанием каждой главы.

- Глава 1 знакомит читателя с понятием, что такое интеллектуальные продукты и объясняет, почему они призваны обеспечить принципиально новый подход к разработке систем.
- Глава 2 содержит высокоуровневое и общее описание системной инженерии.
- В Главе 3 излагается критическая роль требований, которую они играют на протяжении всего жизненного цикла разработки систем.
- Глава 4 показывает как моделирование может улучшить ваше понимание структуры и принципов поведения системы.
- Глава 5 объясняет, как убедиться, что вы не только создаете правильную систему (валидация), но и используете правильные методы при ее создании (верификация).
- Глава 6 содержит описание способов повышения взаимодействия между командами разработчиков.
- Глава 7 познакомит вас с некоторыми реально существующими компаниями, которые инкорпорировали системную инженерию в свою повседневную практику ключевого бизнеса.

Обозначения, используемые в книге

В тексте книги встречаются обозначения и значки, которые должны привлечь внимание читателя к важной информации.

Этот значок предупреждает о важных принципах, которые вам следует запомнить.

Этим значком отмечаются практические советы, которые могут значительно облегчить вашу жизнь.

Это уже не советы, а настоятельные рекомендации. Если их игнорировать, то можно столкнуться с серьезными проблемами и усложнить себе жизнь.

Этим значком обозначается информация, которая дает вам больше подробностей, чем это необходимо, поэтому вы, если хотите, можете ее и пропустить.

Глава 1

Разработка «умных» продуктов

В этой главе

- Спрос на “умные”, взаимосвязанные системы.
- Осознание проблем на пути к успеху.
- Смена стратегии для достижения более обширных целей.

Рассмотрим пример.

Утро. Вы отправляетесь на работу. Пока вы выезжаете на улицу, машина посылает сигнал на включение системы охранной сигнализации в доме и закрытие дверей гаража. Мобильный телефон автоматически синхронизируется с системой голосовых команд вашего автомобиля, а встроенная система спутниковой навигации (GPS) анализирует текущую ситуацию на дорогах и предлагает наиболее выгодный маршрут до работы. Машина сообщает вам о необходимости замены масла и, проанализировав ваше расписание в смартфоне, предлагает удобное время для визита на станцию обслуживания. Более того, из-за прошедшего дождя вы можете получить информацию об ожидаемых затоплениях по маршруту домой.

Разве мы могли когда-либо представить, что «самодвижущийся экипаж» станет для нас таким «умным» помощником? Тем не менее, это так!

В этой главе вы узнаете, какие механизмы лежат в основе интеллектуальных продуктов, как такие продукты взаимодействуют между собой, и почему для их разработки следует внедрять новые бизнес-процессы.

В чем особенность интеллектуальных продуктов?

Сегодня интеллектуальные продукты — неотъемлемая часть нашего существования. Мы с трудом можем представить себе жизнь без программируемой кухонной техники, интерактивных видеоигр, многофункциональных мобильных телефонов, с помощью которых мы записываем видео, фотографируем, просматриваем страницы в интернете и слушаем музыку. Благодаря интеллектуальным продуктам самолеты прокладывают безопасный маршрут, избегая столкновений, а мы можем доверить свою безопасность беспилотным летательным аппаратам и прочим высокоточным системам обороны.

Так что же это за сила, которая наполняет эти и многие другие неживые предметы возможностью проделывать такие удивительные вещи?

Эффективность интеллектуальных систем

«Умные», программно-управляемые системы — ключевой элемент нашей экосистемы:

- **Здравоохранение.** Специальное программное обеспечение позволяет просматривать на мобильном телефоне рентгено- и томографические снимки и результаты анализов. Врач, даже находясь в пути, может быстро проанализировать состояние пациента и на основании этих данных поставить диагноз.
- **Коммунальные услуги.** «Интеллектуальная сеть», обеспечивающая двустороннюю связь между поставщиками электроэнергии и потребителями, повышает эффективность контроля за использованием энергоресурсов. В частности, с помощью этой системы можно запрограммировать включение стиральной машины в период минимальной стоимости электронергии или, наоборот, настроить некоторые приборы на отключение в период пиковых нагрузок. К этой категории можно также отнести «умные» автомобили.
- **Интеллектуальная техника.** Мы можем отслеживать статус энергопотребления домашних приборов благодаря интуитивно понятному интерфейсу. Функция удаленного управления приборами обеспечивает желаемый уровень комфорта и снижение энергозатрат.
- **Развлечения.** «Умные» телевизоры обеспечивают беспроводной доступ к интернету, благодаря чему пользователи могут смотреть фильмы, делать покупки, узнавать о погоде, настраивать новостные страницы и загружать приложения.
- **Транспортные средства.** Интеллектуальные системы предупреждения столкновений интегрируют в себе различные технологии GPS и беспроводного соединения, а также встроенные графические дисплеи, которые предупреждают водителя о соседних транспортных средствах, и даже могут осуществлять экстренное маневрирование.

Смешение технологических ингредиентов

Современные интеллектуальные продукты и сервисы — это результат конвергенции производства, электроники и информационных технологий. Производители быстро поняли, что с помощью новейших достижений в микроэлектронике, программном обеспечении, механических, сенсорных и исполнительных устройствах можно получить большие преимущества и создать продукты, которые приведут в восторг потребителей и нанесут сокрушительный удар конкурентам. Урвав понемногу отовсюду и воспользовавшись знаниями и опытом своих друзей-инженеров, они — *вуаля!* — создали продукты, инновационности которых позавидовал бы сам Герберт Уэллс!

Интеллектуальные продукты могут быть представлены разными формами и размерами, но для них всех характерны следующие три особенности:

- **Измерение и контроль:** Интеллектуальные продукты для спорта, такие как фотокамеры, датчики движения и положения, беспроводные приемники, датчики уровня шума, тепла и влажности, а также магнитных полей — постоянно отслеживают не только работоспособность самого оборудования, но и контролируют условия окружающей среды. Таким образом, можно говорить о том, что интеллектуальные продукты могут в реальном времени адаптироваться к существующим условиям.
- **Взаимосвязь:** Возможности устройств, которые могут взаимодействовать и обмениваться данными между собой, гораздо больше, чем потенциал каждого из них в отдельности. Подключите их к интернету, офисным или другим ИТ-системам — и они сделают невозможное!
- **Интеллект:** Используя сигналы датчиков, анализируя данные о событиях в прошлом и характеристики пользователя, эффективные устройства могут формировать прогнозы, выбирать оптимальные варианты и регулировать работу системы в соответствии с предпочтениями пользователя.

В комбинации с программным обеспечением

Несомненно, что самым важным фактором, который влияет на создание интеллектуальных продуктов, является феноменальный рост вычислительных возможностей. У каждой «умной» системы есть мозг: это встроенные микропроцессоры, которые обрабатывают алгоритмы, анализируют данные, выполняют сложнейшие вычисления и контролируют механические и электронные компоненты умного устройства.

Для выполнения задач им приходится обрабатывать тысячи, а иногда и миллионы строк программного кода. Например, новейшие модели автомашин оснащены десятками микропроцессоров, обрабатывающих до 100 миллионов строк кода, чтобы выполнить от 250 до 300 функций, обслуживающих водителя и пассажиров.

По мере появления все более и более мощных микропроцессоров перед разработчиками программного обеспечения открываются безграничные возможности для воплощения инновационных идей. И это замечательно! - хотя бы потому, что разница в аппаратном обеспечении, которым поначалу могут отличаться продукты, достаточно быстро нивелируется, и продукты быстро теряют такую индивидуальность. Вот почему все чаще отличительные особенности продукта определяются не электронной начинкой, а именно его программным обеспечением, что позволяет продукту завоевывать рынок и становиться лидером.

Благодаря универсальности программного обеспечения производители могут бесконечно разрабатывать дополнительные функции и свойства, а также обновлять продукцию в соответствии с растущими потребностями пользователей. Например, новейшие модели MP3-плееров разработаны далеко не только для прослушивания музыки: в них можно сохранять музыкальные библиотеки, просматривать видео, отправлять и принимать сообщения, играть и использовать сторонние приложения. Лучший продукты — это обновляемые устройства, позволяющие легко добавить новые функции и свойства, чтобы соответствовать изменениям рынка.

Открытые стандарты как необходимое условие распространения

В 2013 году количество потребительских электронных устройств должно приблизиться к ошеломляющей отметке в 1,2 миллиарда. Во всех домах с такими устройствами (а их 800 миллионов) есть доступ к широкополосному интернету. Таким образом, перед поставщиками интернет-решений и услуг открываются безграничные возможности.

А теперь представьте, если бы все электронные приборы создавались в изоляции, если бы каждый производитель разрабатывал свой собственный способ подключения к интернету и обмена данными с внешними устройствами. В эфире стоял бы попросту бесполезный белый шум. Сколько упущенных возможностей!

Но без паники! Умные и дальновидные люди придумали концепцию *открытых стандартов*, т.е. утвержденный и всеми принимаемый способ обмена данными между устройствами и поставщиками услуг. Чем больше компаний используют протоколы связи, соответствующие открытым стандартам, тем больше возможностей открывается перед всеми. Общество все больше эволюционирует

в сторону «интернета вещей» — глобальной экосистемы интеллектуальных и связанных между собой продуктов и услуг.

Ориентированность на покупателя

На сегодняшнй день значительная часть мирового населения состоит из опытных пользователей интеллектуальных продуктов.

Только в 2010 году в мире было продано около 40 миллионов персональных навигационных устройств, а в 2008 — 55 миллионов плееров iPod.

Если вы часто летающий пассажир, то знаете, что вам можно не беспокоиться о плохих погодных условиях: современные воздушные суда оснащены интеллектуальными устройствами, способными обеспечить безопасный перелет. Суть в том, что продвинутые пользователи не только хорошо разбираются в современной «умной» технике, но и формируют спрос на ее новые дополнительные возможности, стимулируя ее дальнейшее развитие.

Пользователям нужны надежные интерактивные устройства, работающие в режиме реального времени, — и нужны не когда-нибудь, а прямо сейчас! Поэтому производителям не стоит расслабляться ни на минуту, - получив продукт, соответствующий последним тенденциям, пользователи очень скоро захотят получить усовершенствованный.

Воплощая в продукте возможности персонализации и интеграции, производитель создает продукты, обслуживающие индивидуальные запросы пользователя и совместимые с его средой обитания. Все покупатели хотят получить продукт, который сделает их жизнь проще и интереснее, но при этом они обладают своим собственным видением идеального продукта, его преимуществ и недостатков. Покупатели хотят иметь такой «умный» продукт, чтобы его настройка под индивидуальные требования пользователя была такой же простой, как загрузка приложения на смартфон!

Интеллектуальные решения не существуют изолированно

Сегодня уже недостаточно просто создать крутой, инновационный, со множеством функций, но «изолированный» и автономный продукт. Современные интеллектуальные устройства должны уметь общаться между собой.

Возьмем, например, современный автомобиль (см. рис. 1-1). Новые модели оснащены массой сложнейших программных подсистем, которые должны сделать процесс вождения максимально приятным, безопасным и топливосберегающим: антиблокировочные тормоза, система безопасности и предотвращения столкновений, управление

микроклиматом и многое другое. Для разработки интеллектуального автомобиля — воистину «системы систем» — требуется интегрировать механические, электрические и электроннные компоненты, а также правильно внедрить около 100 миллионов строк кода.

Но это еще не все. Интеллектуальный автомобиль также взаимодействует с системами внешними по отношению к нему. Службы, основанные на определении местоположения, системы диагностики транспортного средства, системами зарядки гибридных/электромобилей – это лишь малая часть тех систем в составе большой экосистемы, с которыми автомобиль может взаимодействовать.

Это сложный механизм. Но он и чрезвычайно эффективный! Например, если система безопасности вашего автомобиля поддерживает обмен данными с аварийно-спасательными службами, то в случае аварии сведения с внутренних датчиков автомобиля отправляются в службу оперативного реагирования. На основании этих данных, в частности, силы удара, оператор спасательной службы может выбрать оптимальный способ оказания помощи.

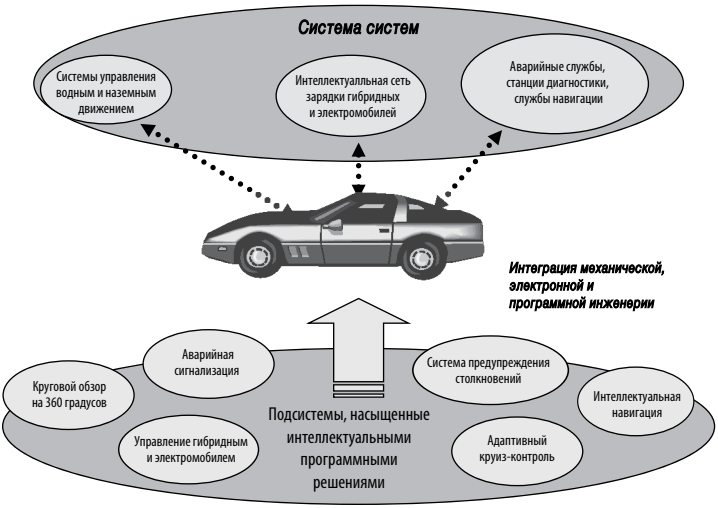

Рис. 1-1: Интеллектуальный автомобиль — это «система систем» в составе еще большей экосистемы.

Ценность интеллектуальных продуктов зачастую в полной мере раскрывается не столько в рамках самого продукта, сколько при его взаимодействии с другими продуктами и услугами в экосистеме. Как и людям, интеллектуальным продуктам необходимо взаимодействовать и обмениваться информацией между собой!

Обмен данными *возможен,* но совсем не *обязателен!* При проектировании систем следует также учитывать аспекты конфиденциальности, безопасности, а также установленные правила.

Старые и новые проблемы

Разработка эффективного интеллектуального продукта в соответствии с постоянно растущими потребностями пользователей — задача не из легких. И далеко не всегда ваш колоссальный опыт в данной области, престижное образование и отчаянное желание являются залогом вашего успеха.

Прежде чем приступить к разработке интеллектуальных продуктов, стоит проанализировать указанные ниже аспекты:

- **Специализация в различных областях.** Вам потребуется экспертиза в различных технических сферах: в производстве, электронике, механике, программной инженерии. Следует заметить, что подобный опыт встречается редко, поскольку большинство компаний специализируются в одной или двух сферах.
- **Разработка ПО мирового уровня.** Если вы относитесь к программному обеспечению как к неизбежному злу, вам стоит начать заняться самогипнозом, чтобы изменить свою точку зрения, т. к. большинство «умного», что есть в интеллектуальных продуктах, зависит именно от него (причем не столько от кода как такового).
- **Совместная разработка аппаратной и программной составляющих.** Поскольку программно-управляемые функции приобретают ключевое значение, разработчики программного и аппаратного обеспечения должны реально вместе создавать продукт, а не обмениваться готовыми модулями, которые впоследствии нужно интегрировать и тестировать.
- **Эффективное управление распределенными командами.** Если команды разработчиков находятся в разных городах, часовых поясах или даже странах, организуйте им все условия для совместной эффективной работы и достижения намеченных результатов.
- **Взаимодействие с другими системами.** Не стоит тратить силы на разработку пусть даже и уникального продукта, но который не сможет коммуницировать с интернетом, офисными ИТ-системам или другими смежными системам. Автономные продукты сложно назвать интеллектуальными и «умными», и они вряд ли будут пользоваться популярностью.

- **Соответствие нормативам.** Даже если ваш продукт не регулируется нормативными или отраслевыми стандартами, он может взаимодействовать с продуктами или услугами, к которым применяются требования таких стандартов. Не игнорируйте этот аспект — и вы всегда будете в выигрыше.
- **Сдвиг приоритетов при проектировании систем.** Делайте ставку не на стабильность и неизменность продукта, а на возможность его дальнейшего усовершенствования. Такой подход обеспечивает наличие *правильных* приоритетов при проектировании. Для авиационной отрасли это может быть, например, безопасность, для оборонной — надежность и защищенность, для производителей бытовой электроники — возможность модернизации.
- **Сокращение срока эксплуатации.** Постоянный спрос на новые функции сокращает срок эксплуатации и быстро делает морально устаревшими даже самые удачные продукты. Ваша задача — выйти на рынок в нужное время и предложить самый востребованный набор функций.
- **Адаптируемость к постоянно меняющимся требованиям.** Как разработчик продукции, вы должны постоянно приспосабливаться к меняющимся требованиям пользователей, динамичной рыночной и конкурентной среде и в соответствии с ними корректировать курс развития компании. Включите интеллект — научитесь получать выгоду от изменений.

Исследуя потенциальные опасности

Игнорируя или незамечая подводные камни при создании новейшего интеллектуального продукта, вы рискуете сесть на мель, не говоря уже о поблекшем имидже, потере репутации и прибыли.

Ошибки, которые кажутся мелкими и легко устранимыми в отдельном продукте, в комплексной системе выглядят гораздо серьезнее, поскольку разбросаны по всей системе. Программная ошибка, не обнаруженная в начале процесса разработки, может нанести вам серьезный ущерб, приведя к возросшей стоимости проекта и несоблюдению его сроков. Поскольку объем программного обеспечения в различных устройствах ежегодно увеличивается вдвое, то становится неудивительным, почему 66% программ не вписываются в бюджет, а 24% крупных проектов закрываются по причине неустранимого отставания от графика.

Если вы не можете разработать сложные продукты в кратчайшие сроки и без ухудшения качества, вы рискуете потерять прибыль и опозорить имя компании. С другой стороны интеллектуальные и взаимосвязанные продукты должны реализовать сотни — а

иногда и тысячи — уникальных требований. Поэтому трудно себе даже представить как можно, не ухудшая качества, сократить цикл разработки.

Если вы не можете оперативно реагировать на новые потребности рынка или угрозу со стороны конкурентов, то не удивляйтесь, что ваше место на рынке начнут занимать более шустрые компании. Для многих организаций разработать исходное проектное решение — это достаточно сложная задача: почти треть всех современных производимых устройств просто не соответствуют эксплуатационным или функциональным требованиям. Не сомневайтесь, все они скоро будут вытеснены более эффективными аналогами.

Наличие у вас первоклассного технического опыта и тщательная и скрупулезная работа над дизайном системы не обязательно гарантируют успех. Разработка многих систем быстро заходила в тупик из-за элементарного несоблюдения требований, из-за проблем в интерфейсах между подсистемами, или из-за слабого взаимодействия и обмена знаниями, т.е. не по причинам, связанным с инженерией.

Осознание необходимости смены парадигмы

Если вы принимаете тот факт, что для достижения успеха в текущих рыночных условиях вам необходимо изменить подход к формированию отличительных ценностей, которые присущи вашим продуктам, вы начинаете пересматривать стратегии планирования, управления и процессы разработки. И сразу же осознаете необходимость смещения фокуса с фактора стоимости на инновационность и перемены — с использованием программного обеспечения, как ключевой основой отличий.

Раньше, когда аппаратные средства играли главенствующую роль, трехмерное автоматизированное проектирование с использованием CAD и спецификация материалов изделия (BOM, bill-of-material) были технологической вершиной в последовательной разработке продукта (см. рис. 1-2). Команда разработчиков аппаратного обеспечения использовала системы CAD для создания аппаратного обеспечения, отвечающего определенным требованиям, в то время как разработчики программного обеспечения работали с иным, но, имеющими связь с вышеупомянутыми, набором требований к созданию необходимого кода. Затем чертежи CAD и спецификации изделий (BOM) передавались в производство, находившее самый быстрый и экономный способ выпуска продукции. Наконец, специалисты по интеграции и тестированию (из другого подразделения) «заливали» программное обеспечение в аппаратное и тестировали изготовленный продукт.

Рис. 1-2: Последовательная разработка продукта.

В современных условиях такой последовательный и жестко регламентированный процесс разработки просто нежизнеспособен. Представьте, что вам необходимо быстро отреагировать на неожиданные рыночные изменения или новые требования к изделию. Каждое усовершенствование потребует возврата к начальной стадии и выполнения всего последовательного процесса заново. Ваш бизнес мгновенно придет в упадок.

Если успех вашей деятельности зависит от интеллектуальных систем, нужно уходить от последовательных жестко регламентированных процессов прошлого и разрабатывать совершенно новые ключевые бизнес-процессы, которые обеспечат ваше процветание в будущем.

Глава 2

Дрессировка тигра методами системной инженерии

В этой главе

- Оценивая возможности системной инженерии
- Приводя в соответствие проектную эйфорию и реалии жизни
- Моделирование дизайна

Вы когда-нибудь задумывались над тем, каким образом NASA удалось создать такую сложную систему, как космический корабль "Аполлон"? Как различные команды конструкторов, программистов, сторонних производителей и других участников смогли коллективно обеспечить первый пилотируемый полет на Луну? Что ж, этого точно бы не случилось без помощи системной инженерии.

Системная инженерия — это междисциплинарный подход к созданию крупных комплексных систем, которые соответствуют определенному набору экономических и технических требований. В аэрокосмической и оборонной промышленности системная инженерия используется уже давно, и многие из полученных опытным путем знаний применяются в других сферах. Поскольку транспортные системы, энергетические, телефонные и сетевые системы становятся все "умнее", для их производства требуются соответствующие технологии эпохи покорения космоса.

Системная инженерия начала применяться в 1940-х годах, но за рамки использования в проектах NASA она вышла только в 1990-х. Именно тогда производители начали превращать обычные изделия в интеллектуальные системы с помощью информационных технологий, и это стало переломным моментом в подходах кразработке продукции. И лишь только с недавних пор ведущую рольво всем этом стало играть программное обеспечение.

Существенно расширив функциональные возможности продуктов и создав многие новые формы взаимодействия между их

компонентами, а также между самими изделиями и окружающим миром, программное обеспечение заняло ключевые позиции в различных продуктах. Новые формы взаимодействия подразумевают экспоненциальное усложнение самих систем, так что у многих компаний не остается выбора, кроме как отказаться от прежних малоуправляемых и неповоротливых процессов разработки и искать новые пути решения задачи по управлению сложностью.

Из этой главы вы узнаете, что такое системная инженерия, как она помогает справляться со сложностями и способствует созданию более интеллектуальных продуктов, делая ваш путь к успеху инновационным.

Знакомство с системной инженерией

Если бы вы спросили у пятерых экспертов, что такое системная инженерия, то возможно получили бы пять - или даже шесть! - различных ответов. Отчасти это обусловлено тем, что термин *системная инженерия* используется для обозначения нескольких различных понятий, а также тем, что сам термин *система* трансформировался десятилетиями, так что значению термина *системная инженерия* ничего не остается, кроме как также видоизменяться с ходом времени.

Но не стоит волноваться. Несмотря на то, что некоторые обладатели ученых степеней могут яростно спорить по поводу тонкостей и масштабов *системной инженерии*, большинство экспертов сходятся во мнении относительно ее основы. В этой главе мы узнаем, что это за основа и как ее использовать.

Увидеть лес за деревьями

Системная инженерия — это одновременно и методика, и процесс (см. рис 2-1):

- Как методика, она охватывает широкий спектр тем: как функционирует система, как она себя ведет в целом, как она взаимодействует с пользователями и другими системами, как коммуницируют ее подсистемы, каким образом объединяются различные технические дисциплины для их совместной деятельности.
- Как процесс, она предлагает четкий структурированный подход к проектированию систем, который применим как на уровне “системы систем”, так и в рамках конкретных технических дисциплин.

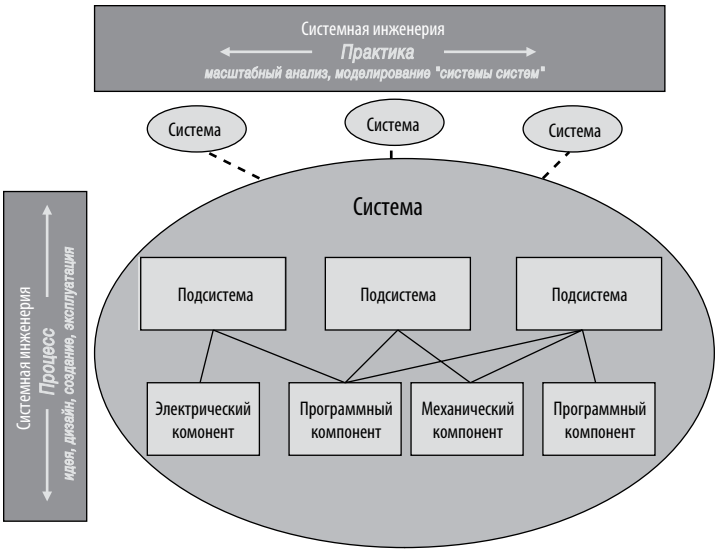

Рис. 2-1: Системная инженерия — это единство методики и процесса.

Как бы там ни было, системная инженерия связана с фактом применения определенной дисциплины в процессе разработки систем. Эта дисциплина имеет две определенные разновидности:

- **Техническая дисциплина** подразумевает неукоснительное соблюдение вами практических процессов разработки— от идеи через производство к последующей эксплуатации.
- **Дисциплина управления** организует и координирует технические усилия на протяжении всего жизненного цикла системы, включая повышение взаимодействия, определение технологических процессов и использование инструментов разработки.

Следование некоторым ключевым принципам

Проникая вширь и вглубь проекта, системная инженерия помогает вам контролировать детали сложного и многостороннего процесса разработки, при этом не позволяя упускать из вида глобальные цели проекта.

Вам будет проще, если вы начнете придерживаться следующих ключевых принципов:

- **Стремиться к цели.** Определите желаемый результат в самом начале проекта и не сходите с пути его достижения, как бы ни складывались обстоятельства и какими бы неожиданными ни были промежуточные результаты.
- **Привлекать все заинтересованные стороны.** Пользуйтесь информацией, получаемой от ваших заказчиков, пользователей системы, операторов, руководителей высшего звена и многих других, при принятии решений на различных стадиях процесса разработки.
- **Формулировать задачи до принятия способа их решения.** Придерживайтесь принципа открытости мышления – это поможет вам получить различные варианты решения задачи, и вы наверняка найдете именно то, которое наилучшим образом будет соответствовать поставленной цели.
- **Разбивать сложные задачи на составляющие, с которыми легче справиться.** Решение задачи значительно упроститься, если вы декомпозируете вашу систему на менее масштабные подсистемы, а затем разделите каждую подсистему на аппаратные или программные компоненты. Другим схожим принципом является управление интерфейсами между компонентами для обеспечения их полноценной интеграции и требуемой итоговой эффективности.
- **Не спешить с принятием узкоспециальных технологических решений.** Дождитесь наступления полной ясности в процессе разработки перед тем, как начать выбор физических компонентов, чтобы не допустить использования в новом продукте устаревших или уже ненужных технологий на момент начала реализации вашего дизайна.
- **Обеспечить связь между требованиями и дизайном.** Будьте уверены в том, что вы всегда сможете обосновать ваши проектные решения, обеспечив их обратную связь с соответствующими техническим и экономическим нуждами заказчика.
- **Тестируйте рано, тестируйте постоянно.** Используйте прототипы, симуляторы, эмуляторы и любые иные средства для того, чтобы каждый, кто вовлечен в разработку, как можно раньше мог ознакомиться с создаваемой системой. Результаты тестов должны быть обязательно связаны с требованиями, чтобы подтверждать, что вы неукоснительно соблюдаете их.

Изучение процесса системной инженерии.

Хорошо иметь под рукой руководящие правила и инструкции, но как применить их на практике и заставить работать? Единственным верным выходом из положения становится разработка согласованного процесса системной инженерии, в котором все они будут воплощены.

За последние 20 лет или около того эксперты в области разработки сложных систем создали и усовершенствовали то, что нынче известно как *V-модель* процесса системной инженерии (см. рис. 2-2). Эта модель является графическим отображением набора шагов и процедур, используемых при разработке сложных систем.

Рис. 2-2: V-модель процесса системной инженерии.

Перемещаясь по V-модели слева направо, вы реализуете процесс системной инженерии в виде серии следующих шагов:

- **Концепция использования (ConOps):** Определите и задокументируйте потребности основных заинтересованных сторон, общие возможности системы, роли и обязанности, а также измеряемые показатели эффективности, которым система должна соответствовать при ее валидации по окончании проекта.
- **Характеристики системы:** Разработайте набор поддающихся проверке системных требований, которые отвечают нуждам

заинтересованных сторон, определенным на стадии разработки и принятия концепции использования системы.

- **Высокоуровневое проектирование:** Разработайте высокоуровневую архитектуру, которая удовлетворяет системным требованиями, обеспечивает обслуживание, возможную модернизацию, а также интеграцию с другими системами.
- **Проектирование компонентов:** Постепенно детализируйте системный дизайн, формируя такие требования к компонентам, которые позволят приобретать и использовать аппаратные средства, стоимость которых не будет выходить за рамки бюджета.
- **Разработка программного обеспечения / аппаратных средств:** Выбирайте и закупайте соответствующие технологии. Разрабатывайте программное обеспечения и аппаратные средства, чтобы соответствовать детализированным спецификациям вашего дизайна.
- **Тестирование модулей/устройств:** Тестируйте реализацию каждого аппаратного компонента вашего изделия, верифицируя его функциональность на соответствие требованиям этого уровня.
- **Тестирование подсистем:** Интегрируйте аппаратные и программные компоненты в подсистемы. Тестируйте и верифицируйте соответствие каждой подсистемы высокоуровневым требованиям.
- **Тестирование системы:** Интегрируйте подсистемы и тестируйте всю систему целиком на предмет ее соответствия системным требованиям. Верифицируйте, все ли интерфейсы были корректно реализованы и все ли требования и ограничения были соблюдены.
- **Приемочные испытания:** Валидируйте соответствие системы поставленным требованиям и ее эффективность в достижении запланированных целей.

На протяжении всего процесса системной инженерии вы разрабатываете и совершенствуете системную документацию. На каждом шаге в левой части V-модели (см. рис. 2-2) вы создаете требования, формирующие и определяющие следующий шаг, а также план верификации реализации текущего уровня декомпозиции.

Например, на стадии разработки концепции использования (ConOps) вы создаете документ, содержащий высокоуровневые требования, которые, с одной стороны, определяют спецификацию системы и ее характеристики, а с другой стороны, - помогают вам создать *План валидации системы*, который, в свою очередь, обуславливает и определяет приемочные испытания. На каждом шаге в правой части V-модели создается документация по обучению работе с системой, ее обслуживанию, установке и тестированию.

Сопрягая все шаги в левой части V-модели посредством требований и обращаясь к этим требованиям каждый раз, когда вы перемещаетесь по правой части V-модели, вы можете быть уверены, что в большей степени придерживаетесь изначально выбранной миссии и сохраняете объективность на протяжении всего процесса. Это сопряжение и взаимосвязь в системной инженерии характеризуется понятием *трассируемость*. Глава 3 описывает требования и трассируемость более подробно.

Управление сложностью с помощью моделей

Использование некоторых моделей весьма полезно, в особенности при работе над сложным инженерным проектом. Если вы сможете разработать относительно недорогие методы проектирования, тестирования и верификации системы *до* ее непосредственного создания, то вы сохраните массу средств и времени, - а может быть даже и свое рабочее место! Для этого при проектировании и совершенствовании системы на протяжении всего процесса разработки важно использовать системное моделирование.

Модели системы дают вам возможность собирать и фиксировать сложности на самых различных уровнях, включая уровень “системы систем” (также известный как уровень экосистемы), уровни системы, подсистем и компонентов. Модели позволяют вам исследовать детали и нюансы сложностей на каждом из этих уровней (так называемых *уровнях абстракции*) независимо друг от друга, и при необходимости либо скрывать их, либо, наоборот, особо подчеркивать их.

Модели могут иметь различные формы. На простейшем уровне это может быть обычная электронная таблица для вычисления какой-либо эмпирической характеристики системы. С другой стороны, это может выглядеть как сложнейший интерактивный компьютерный симулятор.

Допустим, вам необходимо понять, как автомобильная система - или, проще говоря, автомобиль, который вы разрабатываете, - будет фиксировать и пересылать данные об аварийном столкновении внешней системе быстрого реагирования. Для этого вам нужно будет изучить информацию о соответствующих автомобильных датчиках, о том, как автомобиль взаимодействует с внешней системой, и т.д. Однако вы не желаете вникать в лишние детали и изучать кабельную разводку автомобиля или схему расположения компонентов. В этом случае правильно сконструированная модель поможет вам получить необходимые данные без излишней скрупулезности.

Модели дают вам следующие преимущества:

- Они позволяют вам сконцентрироваться исключительно на значимой информации рассматриваемого вопроса, но при этом обеспечивают соблюдение целостности и логичности всего вашего дизайна.
- Они отображают как структуру (архитектуру), так и поведение (функциональные возможности) системы, иллюстрируя взаимоотношения и взаимодействия между ее элементами.
- Вы можете использовать модели для прогнозирования поведения системы на различных уровнях абстракции, изучая альтернативные варианты построения архитектуры на ранних стадиях процесса разработки, а также исследовать уровень затрат для выбора наиболее подходящего проектного решения.
- Вы можете разрабатывать исполняемые модели, которые позволят провалидировать ваш дизайн на соответствие предъявляемым требованиям еще до начала построения системы.
- Модели позволяют рассмотреть особенности системы с различных точек зрения (архитектурной, логической, функциональной, материальной, информационной, пользовательской), что способствует выявлению специфических проблем.
- Вы можете использовать модели для управления процессом разработки.
- Модели дают абсолютную ясность и прозрачность системы, тем самым направляя ее обсуждение в русло полного взаимопонимания.
- Модели предоставляют возможность обмена идеями и выработки критериев принятия решений, делая ваше взаимодействие и сотрудничество с командой разработчиков более эффективным.
- И что наиболее важно, - модель системы позволяет синхронизировать множество инженерных дисциплин, тем самым решая одну из самых сложных задач, возникающую при разработке интеллектуальных систем, - координацию разработки аппаратных и программных средств.

Говорим на одном языке

Также как архитектор использует в строительном чертеже стандартный набор архитектурных элементов, которые будут понятны любому руководителю строительства, так и команда разработчиков должна использовать универсальный общеупотребительный подход при описании модели системы, чтобы обеспечить ее ясность и общее понимание.

В 2001 году Международный совет по системной инженерии (International Council on Systems Engineering, INCOSE) совместно с Консорциумом по объектно-ориентированным технологиям (Object Management Group, OMG) выступили с инициативой разработки общего языка моделирования для системной инженерии, и через несколько лет был создан язык системного моделирования (Systems Modeling Language, SysML). Разработанный на базе унифицированного языка моделирования для разработки программного обеспечения (Unified Modeling Language, UML), SysML де-факто стал стандартным языком моделирования систем и "системы систем".

В SysML используется простой схематичный подход к моделированию систем (настолько простой, что некоторые называют его «наскальной живописью»), в котором основная единица структуры — блок — может означать аппаратное средство, программное обеспечение, устройство, персонал и любые другие элементы системы. С помощью серии вложенных *структурных диаграмм* вы определяете внутреннюю структуру и предполагаемое использование конкретного элемента системы (например, системы АБС автомашины). Затем с помощью отдельной серии вложенных *диаграмм поведения* вы иллюстрируете взаимодействие этого элемента с остальными элементами системы и с *действующими лицами* (пользователями, внешними системами или окружением – actors) для выполнения или *реализации* такого поведения.

Кроме моделирования структуры (архитектуры) и динамического поведения (функционирования) системы, SysML также позволяет моделировать требования и эксплуатационные показатели. Например для автомобильной системы вы можете создать диаграмму требований, чтобы обозначить ограничение ("тормозной путь на скорости 100 км/ч на чистой, сухой поверхности не должен превышать 55 метров"), а также параметрические диаграммы, которые описывают формулы, определяющие движение автомобиля. И самое интересное — вы можете воспользоваться новыми программными инструментами, например, для создания симулятора нового спортивного автомобиля. Это означает, что вы можете взять автомобиль на тест-драйв и проверить его управляемость еще до того, как он будет построен.

Определяя и организуя конструктивные элементы модели, SysML заставляет системных инженеров и архитекторов быть более ясными и точными при разработке системы. Такой подход устраняет двусмысленность и обеспечивает более высокое качество, сокращает циклы разработки и снижает издержки.

Лучше один раз увидеть...

Лучший способ почувствовать, что из себя представляет моделирование с использованием SysML, — это взглянуть на диаграмму SysML. На рис. 2-3 изображена простая *контекстная диаграмма* современного автомобиля со встроенной навигационной системой GPS и дистанционным управлением системой безопасности.

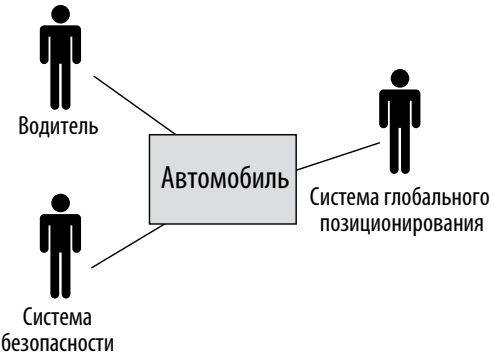

Рис. 2-3: Простая контекстная диаграмма автомобиля

Контекстная диаграмма применяется для определения границ (или контекста) использования системы. В данном случае системой является автомобиль, который взаимодействует с тремя внешними *действующими лицами*: водителем, системой GPS и системой безопасности. Действующее лицо — это любой объект, с которым взаимодействует система, будь то пользователь, другая система или окружение. Вы используете контекстную диаграмму для обозначения масштаба разрабатываемой системы.

Определяя систему и ее действующие лица, вы тем самым обозначаете взаимоотношения между ними, а следовательно формируете системные требования и интерфейсы. Затем можно начинать составлять спецификации интерфейсов и определять потоки данных между системой и ее элементами.

Без контекстной диаграммы вы можете упустить из виду одно или даже два действующих лица, и в результате установленные вами системные требования окажутся неполными. Например, если вы забудете присвоить системе безопасности статус действующего лица в экосистеме вашего автомобиля, то каким бы умным ни был автомобиль, он не сможет поддерживать систему безопасности.

Глава 3

Революционизация требований

В этой главе

- Стоит признать, что изменение – это не так уж и плохо
- Как сформировать полную базу требований с помощью прецедентов использования
- Последовательная связь требований в процессе разработки
- Анализ влияния изменений
- Учимся управлять требованиями

В старые добрые времена требования разрабатывались в самом начале проекта, и проектирование изделия велось в строгом соответствии с ними. Так что, если заказчику вздумывалось внести пару-тройку изменений, вы - как разработчик - начинали топать ногами и извергать тирады, жалуясь на утомительный процесс согласования изменений в проектной документации, который потребует заново собрать 57 подписей.

Теперь так нельзя. В условиях меняющегося рынка и жестокой конкуренции, когда успех как никогда зависит от удовлетворения нужд клиентов, у вас есть всего два пути: либо меняться, либо оплакивать поражение. Из этой главы вы узнаете, как разрабатывать системные требования, помня о необходимости их возможного изменения, и как обеспечивать соответствие ваших проектных решений этим требованиям.

Философия изменений

В новый век "поумневших" товаров вам приходится оперативно реагировать на динамику рынка, - меняющиеся потребности заказчиков, достижения конкурентов или законодательные нововведения. Разработчики уже годами отмечают следующую тенденцию — требования к продукту *должны* меняться по мере того, как в процессе разработки вы начинаете лучше понимать нужды заказчика. Но при традиционном подходе к разработке систем, изменения - это враг. Так что же делать в этой ситуации?

Стоит задуматься о внедрении нового процесса.

Возможно вас обрадует тот факт, что до вас этот путь уже был пройден другими разработчиками, создавшими абсолютно новый процесс *инженерии требований*. Состоящий не столько из методов предварительного анализа и формирования требований, инженерия требований последовательно от начала и до конца определяет процесс разработки требований, привязке их к тестам, значительно облегчает внесение изменений.

Требования работают более эффективно, если при разработке в них закладывается возможность изменений. В основе философии инженерии требований лежит тот факт, что изменения всегда желанны. По сути изменение — это и есть цель!

Понимание контекста

Перед тем, как вы начнете разрабатывать набор требований для интеллектуального продукта или системы, вам приходится уделять некоторое время обдумыванию проблем и задач, которые этот продукт или система будут решать. Например, если вы собираетесь создать автомобиль, важно понять, как именно он будет использоваться и кем. Станет ли он ездить по городу, магистралям или раллийным трассам? Кто является целевой аудиторией этого автомобиля на рынке — опытные водители, молодежь или их родители? Будет ли он эксплуатироваться в жестких условиях (например, в условиях севера), на посыпанных солью дорогах, в экстремально жарком климате, на пересеченной местности или на высокогорье?

Понимание *контекста* критически важно при разработке систем, которые должны достичь экономических и маркетинговых целей. Под контекстом мы подразумеваем совокупность *действующих лиц* (пользователей, другие системы, окружение), с которыми

наша система взаимодействует, а также то, как происходит такое взаимодействие.

На рис. 3-1 изображена контекстная диаграмма автомобиля со встроенной навигационной системой и дистанционным управлением системой безопасности, который предназначен для использования в холодном климате. Автомобиль на рисунке представлен в виде "черного ящика", который взаимодействует со следующими действующими лицами: водителем, навигацией GPS, системой безопасности и холодной окружающей средой. Определение контекста помогает понять, какие функциональные возможности необходимы системе, и какой обмен данными будет происходить между ней и ее элементами.

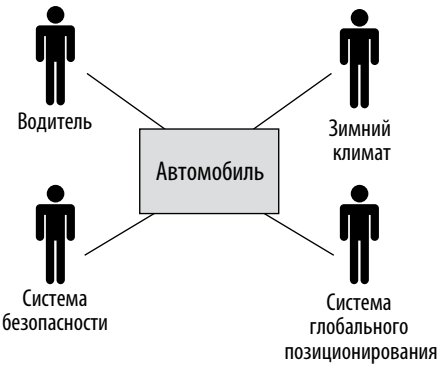

Рис. 3-1: Контекст системы очерчивает границы и определяет интерфейсы.

Понимание контекста дает вам уверенность в том, что все необходимые требования, связи и интерфейсы были учтены до начала процесса разработки. К примеру, если для автомобильной системы не будет назначено действующее лицо "зимний климат", вы не сможете определить технические требования для "зимнего пакета опций", в который входят усиленные провода от аккумуляторной батареи и дополнительное защитное покрытие кузова.

На самом высоком уровне абстракции ваша система представляет из себя "черный ящик", взаимодействующий с различными внешними элементами. Если вы заглянете внутрь этого ящика, то обнаружите ряд взаимосвязанных подсистем (например, антиблокировочную систему, навигацию и т.д.), которые в совокупности образуют общую систему. В свою очередь каждая из подсистем может декомпозироваться на ряд взаимосвязанных компонентов (или "подподсистем"). И при этом на каждом *уровне декомпозиции* (система, подсистема, компонент) внутри вашей системы происходит изменение контекста.

Сравните контекстную диаграмму автомобиля (см. рис. 3-1) с контекстной диаграммой встроенной в него навигационной подсистемы (см. рис. 3-2). На уровне системы автомобиль является черным ящиком, взаимодействующим с внешними элементами. На уровне подсистемы, черным ящиком является навигационная подсистема, и она взаимодействует с другим набором элементов (действующих лиц): водителем, подсистемой электрики и системой глобального позиционирования GPS.

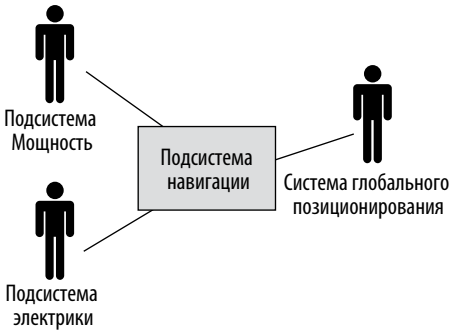

Рис. 3-2: Контекст изменяется при переходе от одного уровня системы к другому.

Понимание контекста позволяет устанавливать границы и определять интерфейсы системы, таким образом способствуя выработке четких системных требований.

Погружение в мир требований

Требования можно разбить на три категории (или класса):

- **Исходные требования.** Это такие требования, которые вы получаете от заказчиков или других заинтересованных сторон. Они могут быть пространными и общими, детализированными и специфическими, полными и фрагментированными, - в большинстве случаев, в них есть всего понемногу. Как принято говорить, *"когда дело доходит до требований, заказчик не приемлет никаких правил – вы получаете то, что получаете"*.
- **Целевые или бизнес требования.** Это требования, определяющие контекст, в котором система будет функционировать, — но не то, что именно она делает, а ее роль, которую она играет в глобальном окружении. Скажем, для нового военного самолета это будет описание миссий, для которых он будет предназначаться. Для нового смартфона

бизнес требования могут определять, каким образом он будет функционировать в инфраструктуре конкретного мобильного оператора.

- **Системные/подсистемные требования.** Эти требования определяют то, что система должна иметь возможность делать. Они берут свое начало на высоком системном уровне, затем анализируются и декомпозируются, чтобы образовать требования для подсистем уровнем ниже. Они могут выражаться в простых формах ("система должна..."), либо изображаться в виде моделей и диаграмм.

На каждом уровне абстракции требования определяют, *ЧТО* система должна делать и *КАК* – насколько хорошо - ей следует это делать, но никак не то, как это должно быть реализовано. С учетом растущей сложности современных интеллектуальных систем становится практически невозможным четко сформировать системные требования в самом начале процесса разработки. Например, в таких областях, как бытовая электроника, где изменения важны больше, чем стабильность, вы вряд ли станете на самом старте проектирования системы "замораживать" ваши требования.

Поиск дополнительной информации

На ранней стадии процесса инженерии требований важно собрать как можно больше данных, касающихся ваших исходных требований, получив их от максимально возможного числа заинтересованных сторон. Понятно, что сбор информации вы начинаете со своих заказчиков, однако стоит обратить внимание на стандарты в области законодательства, промышленности и безопасности производства, которые оказывают влияние на вашу систему, ее интерфейсы и обмен данными с другими системами (например, GPS), а также учесть экономические и маркетинговые ограничения.

Самым лучшим результатом ваших трудов станет набор требований, которые описывают возможности системы, а не ее функции. (Т.е. то, *что* должна делать система, а не то, *как* она должна функционировать). Если заказчик описывает определенную функцию, то следует уточнить, зачем она ему нужна. Таким образом вы сможете понять, о каких возможностях системы идет речь.

Также важно определить *функциональные* и *нефункциональные* требования. Функциональные требования описывают то, что должна делать система при определенных входных условиях. К примеру, получив информацию об отправной и конечной точках, ваша навигационная система должна проложить маршрут и отобразить

его на дисплее. Нефункциональные требования используются для описания характеристик производительности и качества, а также фиксации определенных ограничений, которые могут налагаться на дизайн. К таким требованиям относятся скорость, мощность, надежность, вес, удобство и масштабируемость.

Чтобы убедиться, что вы учли все эксплуатационные факторы, вы разрабатываете *прецеденты использования*, которые описывают все возможные разновидности использования функций системы. Например, функция "проложить маршрут" в навигационной системе может использоваться для "поиска ближайшей АЗС" или для "отображения гостиниц на пути следования". Прецеденты использования, как правило, описывают последовательность выполнения одной или нескольких функций системы. Эти сценарии описывают работу системы в конкретных условиях и могут использоваться для всех трех классов требований — исходных, целевых и системных/подсистемных.

При разработке требований всегда задумывайтесь над тем, как реально будет использоваться система, и тогда она станет по-настоящему значимой и полезной.

Извлечение требований

После того, как вы обозначили контекст вашей системы и сформулировали ряд исходных высокоуровневых системных требований, вы можете переходить к процессу проектирования системы на высоком уровне (см. рис. 3-3). В процессе проектирования вы разрабатываете дополнительные требования (например, план верификации системы, который должен быть выполнен во время ее тестирования), или формируете ограничения архитектурного дизайна системы, подлежащие обязательному соблюдению при детальной проработке.

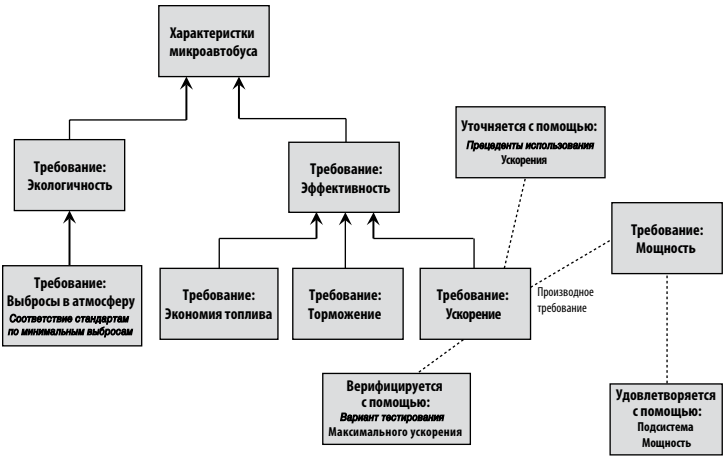

Рис. 3-3: Процесс высокоуровневого проектирования.

Постепенно углубляясь в дизайн, вы обеспечиваете выполнение требований более высокого уровня, а также формируете новые требования для использования их на соответствующем уровне тестирования и последующем (более низком) уровне проектирования.

Как вы поняли, определение самого понятия "требования" имеет тенденцию расширяться, поскольку границы между требованиями и дизайном постепенно размываются, а наиболее важным становится взаимосвязь между тестированием и разработкой. Окончательный набор ваших требований представляет собой комбинацию исходных высокоуровневых системных требований и тех, что получены в процессе разработки.

Создание диаграмм требований

Несмотря на то, что требования в виде старомодных текстовых наборов по-прежнему необходимы при работе над крупными проектами, которые связаны с выполнением контрактных обязательств, они зачастую не отвечают запросам и ожиданиям современного мира со множеством сложных, гибких, ориентированных на потребителя, интеллектуальных продуктов. К счастью, широко используемый язык SysML дает вам прекрасную основу для моделирования требований. В то время как большие наборы текстовых требований могут по-прежнему существовать в базе данных (особенно если они поступают непосредственно от заказчика), тем не менее имеются новые и более удобные

методы визуализации и работы с наборами взаимосвязанных ассоциированных требований.

Язык SysML позволяет создавать иерархические модели требований, которые иллюстрируют взаимосвязи и зависимости, разделяют требования на исходные и производные, предоставляют логическое обоснование тех или иных проектных решений.

На рис. 3-3 отображены два системных требования высокого уровня: Экологичность и Эффективность. Требование "Экологичность" имеет особое дополнительное (вытекающее) требование — "Выбросы в атмосферу". Требование "Эффективность" имеет три дополнительных требования, одно из которых "Ускорение". При этом производное требование "Ускорение" дополнительно *уточняется с помощью* прецедентов использования. Определен план верификации максимального ускорения (который является, в свою очередь, требованием, которое должно удовлетворяться в процессе тестирования). Наконец, дополнительное требование "Ускорение" создает производное требование к мощности, которое *удовлетворяется с помощью* "Подсистемы Мощность".

Диаграммы требований в SysML позволяют с легкостью отображать взаимосвязи между элементами и давать ответы на такие вопросы, как: "Каким требованиям удовлетворяет "Подсистема Мощность"?" или "К каким последствиям приведет изменение параметра "Размер багажного отделения "?". Если относиться к требованиям как к неотъемлемой части системной архитектуры, будет гораздо проще оценить влияние их изменений на вашу систему. Разве это не одна из целей системной инженерии?

Выработка проектных решений на основе требований

По мере того, как ваши требования последовательно «движутся» через процесс проектирования, сущность некоторых требований может обуславливать ваш выбор определенных проектных решений. К примеру, если вы проектируете авиационную систему предупреждения столкновений, то в начале процесса у вас могут возникнуть три варианта проектного решения, каждый из которых будет соответствовать функциональному требованию как избежать столкновений:

- Передовая радиолокационная система на борту
- Система «запрос-ответ»

- Основное ручное управление (плюс связь с авиадиспетчерской службой)

Каждое проектное решение состоит из собственного уникального набора компонентов, требований к размещению, стоимостных характеристик и подразумевает определенное взаимодействие между системой и ее элементами (например, авиадиспетчерской службой). При дальнейшей декомпозиции вашего дизайна вам может встретиться, например, такое нефункциональное требование "... обеспечить максимальное рабочее пространство кабины пилота", которое будет ограничивать ваш проектный выбор (радиолокационная система может просто не поместиться).

Системные инженеры часто используют метод *исследования затрат* для оценки различных проектных решений. По сути это то же, что проделывает каждый из нас перед тем, как сделать важный выбор. Определяем ключевые факторы, влияющие на принятие решения, оцениваем имеющиеся варианты и степень их влияния на эти факторы, добавляем вес каждому фактору и принимаем решение. Следует заметить, что исследования затрат зачастую требуют проведения глубокого и всестороннего изучения и анализа, если требуется определить полную оценку альтернатив.

Следить за выполнением всех требований, а также контролировать их влияние на остальные элементы системы — это те неизбежные обязанности, которыми мы должны заниматься, если хотим создать успешный продукт. Имеющиеся на рынке программные инструменты помогут вам управлять системными требованиями.

Если при разработке вы пропустите какое-либо требование, это может привести к серьезным проблемам. Например, вы можете спроектировать самый лучший в мире автомобильный подстаканник. Но если он будет расположен прямо под автомобильной магнитолой, и ваш двойной кофе закроет доступ к кнопкам управления радио, - поскольку вы не учли требование "... обеспечить допустимый зазор", — вас постигнет неудача. Исправление подобных ошибок может реально дорого обойтись вам.

Непостоянство требований и проектных решений

Допустим, что в примере с предупреждением столкновений в воздухе из предыдущего раздела, в качестве проектного решения вы выбрали систему «запрос-ответ». Эта полуавтоматическая опция подразумевает привлечение к работе авиадиспетчерской

службы, которая должна мониторить ретрансляцию сигналов с борта самолета в зоне его полета. Ваше проектное решение формирует дополнительные (вторичные) требования для разработки подсистемы «запрос-ответ», а также план ее тестирования.

Вы полностью погружаетесь в разработку подсистемы, но вдруг получаете запрос на изменение требования. Новое требование утверждает — самолет должен иметь возможность предупреждать столкновения самостоятельно (независимо от наземных служб). Но ваше проектное решение не соответствует этому требованию, поэтому вы должны отложить свое решение в сторону и начать работу над другим вариантом — радиолокационной системой.

Определенные изменения могут повлечь за собой возникновение дополнительных требований на уровне подсистемы, что в свою очередь может распространиться и на дизайн основной системы. К примеру, финансовые ограничения могут сузить выбор системных компонентов, что, в свою очередь, может сказаться на функциональности подсистем и повлиять на исходные требования к системе. В недавние времена (несколько лет тому назад) такой процесс распространения и учета изменений отнимал много дней и даже недель, так как иженерам приходилось находить и изменять соответствующую документацию и свои проектные решения, существующие в виде текстовых документов, презентационных слайдов и электронных таблиц. Использование моделей в этом процессе существенно упростило ситуацию (более подробно - см. главу 4).

Вам следует создать формальный процесс управления изменениями требований, чтобы вам было проще понимать и оценивать влияние изменений на дизайн разрабатываемой системы.

Сопряжение требований с тестированием

Вы только что закончили проектирование новой сверхчувствительной навигационной системы автомобиля. Был создан ее прототип, и команда тестировщиков докладывает, что система работает стабильно. Она рассчитывает и прокладывает маршрут между двумя точками с такой сумасшедшей скоростью (благодаря разработанному вами потрясающему алгоритму), что у ваших конкурентов просто не остается шансов. Ваш руководитель гордится успехом и предлагает генеральному директору компании попробовать в течение дня как работает прототип. Генеральный директор был впечатлен работой

системы. Но ровно до тех пор, пока ему не захотелось с помощью навигатора найти ближайший ресторан китайской кухни — и вот он уже раздражен долгим ожиданием ответа.

Вы пытаетесь найти причину и выясняется, что встроенный алгоритм был оптимизирован для расчета маршрута от точки до точки, но не для поиска объектов. И хотя согласно плану тестирования было проведено успешное тестирование функциональности всей системы, возможно не все прецеденты использования были покрыты тестами (или кто-то забыл создать такой прецедент использования).

Ключевым компонентом процесса системной инженерии является установление связей между требованиями и тестами. Вы должны быть уверены, что создаете систему именно такой, какой вы ее наметили, а не просто систему, которая будет "работать".

На каждом уровне декомпозиции системы, по мере того, как вы уточняете и совершенствуете требования, вы создаете и совершенствуете планы их тестирования, чтобы верифицировать, что система соответствует этим требованиям. Если требования меняются вы должны поменять планы тестирования. При этом зачастую сам факт создания критериев для тестирования тех или иных требований помогает вам улучшить и усовершенствовать сами эти требования.

Хорошей практикой является оценка ваших требований с точки зрения самой *возможности* их тестирования и даже попытка заранее определить *как* они будут тестироваться.

Все должно отслеживаться

Чтобы убедиться, что все ваши требования реализованы надлежащим образом у вас должна быть возможность отслеживать каждое из них на протяжении всего процесса разработки и тестирования.

Трассируемость требований — это возможность связать каждое из них с тремя взаимосвязанными элементами:

- Потребностями заказчика, которые описываются требованиями (исходными требованиями)
- Элементами системы, которые реализуют требования
- Сценарием тестирования, который верифицирует требования

Трассируемость требований помогает удостовериться в их соответствии нормам и стандартам, не упустить ни одного требования и постоянно фокусироваться на намеченных целях проекта. С помощью сквозной трассируемости вы можете заранее оценить, на

что конкретно повлияет последнее изменение требования или выбор альтернативного проектного решения.

Награда за тяжкий труд

В больших и сложных системах управление требованиями может превратиться в кошмар. Многие команды разработчиков насчитывают сотни (и даже тысячи) архитекторов и инженеров, и все они имеют дело с требованиями — создают, редактируют, проверяют или просто пытаются понять их. Трассируемость, как правило, пронизывает четыре уровня декомпозиции – от требований заказчика, которые последовательно опускаются на уровень разработки компонентов, и до тестирования. Эти же уровни декомпозиции часто накладываются и на границы цепочек поставок, что еще больше усложняет жизнь.

Эффективное управление требованиями охватывает многие инженерные дисциплины, включая системный дизайн, а также архитектурную, программную, механическую, электрическую и тестовую инженерию. Экономические функции, такие как маркетинг и закупки, также осуществляются в процессе управления требованиями.

Программные инструменты могут помочь вам справиться с неповоротливым и громоздким процессом управления требованиями. Они позволяют контролировать историю внесения изменений, сохранять все изменения, проводить анализ влияния изменений на требования и автоматизировать процесс управления изменениями. Они также сообщат вам о пропущенном требовании, технически переусложнённом дизайне, несоблюдении норм и правил.

Управление требованиями — сложная задача, но с помощью подходящих средств и инструментов вы сможете добиться успешного результата, который будет отвечать бюджету, графику, будет работоспособным - и сохранит ваши нервы и здоровье.

Глава 4

Переход к абстракции с помощью системного моделирования

В этой главе

- Декомпозирование системы на уровни абстракции
- Визуализация элементов системы
- Отображение поведения системы
- Итерационное совершенствование системных моделей

Формула — это тоже своего рода модель. В ней запечатлены математические связи между входными переменными и математическая структура, которая применима к широкому спектру входных данных. Визуализация формулы помогает вам лучше понять взаимосвязи между элементами структуры. А использование формулы позволяет вам тестировать полный диапазон самых разных возможностей для поиска оптимального для вас варианта.

Из этой главы вы узнаете, как можно использовать системное моделирование для управления сложностью, как с его помощью выделять самые основные отношения внутри системы, а также проводить экономное тестирование до начала производства.

Моделирование системной архитектуры

Архитектурные модели дают представление о статической природе вашей системы, включая ее структуру и сферу ее применения. Давайте рассмотрим упрощенную архитектурную модель автомобиля, которая приведена на рис. 4-1 и 4-2.

Неполная структурная модель на рис. 4-1 отражает декомпозицию системы на несколько уровней. На самом верхнем уровне представлена система в целом: автомобиль. Уровнем ниже находятся две подсистемы автомобиля: Антиблокировочная тормозная система (АБС) и Шасси.

Подсистема Шасси далее декомпозируется на подсистему Ступица в сборе и механический компонент Шина, а также и на другие подсистемы и компоненты, которые для простоты не показаны на этой диаграмме. В свою очередь, подсистема Ступица в сборе состоит из электронного компонента (Датчик) и нескольких других механических компонентов, которые также не показаны на рисунке.

Подсистема АБС декомпозируется на механический компонент (Ротор) и еще одну подсистему (Антиблокировочный блок управления). При этом подсистема Антиблокировочный блок управления состоит из двух электронных компонентов: Датчика силы сцепления и Регулятора тормозного усилия. Пунктирная линия, соединяющая Датчик Ступицы с Антиблокировочным блоком управления, показывает, что между этими двумя компонентами существует связь, хотя Датчик при этом и не входит в блок управления.

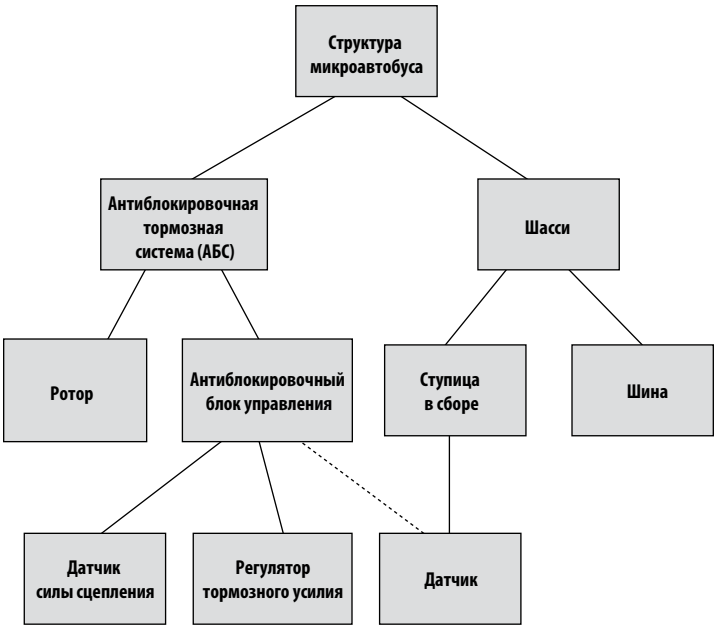

Рис. 4-1: Упрощенная архитектурная модель системы микроавтобуса.

Структурные модели, подобные упрощенной архитектурной модели на рис. 4-1, отображают иерархическую структуру системной архитектуры и показывают взаимосвязи и отношения между элементами модели.

Рис. 4-2 показывает другую часть полной архитектурной модели - это взгляд изнутри на то, как компоненты подсистемы Антиблокировочного блока управления взаимодействуют между собой и с Датчиком Ступицы. В данном случае выход Датчика сту-

пицы соединен со входом Датчика силы сцепления, выход которого, в свою очередь, подключен ко входу Регулятора тормозного усилия. Эта простая схема иллюстрирует лишь связь между элементами, чтобы показать их *предполагаемое* использование, но не отражает условий, при которых они будут взаимодействовать. Модели поведения (о них мы поговорим в следующем разделе) описывают последовательность событий, которые должны наступить, чтобы вызвать взаимодействие элементов (например, Датчик ступицы определяет потерю сцепления, в результате чего Датчик силы сцепления приводит в действие Регулятор тормозного усилия).

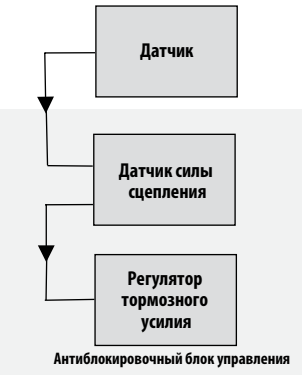

Рис. 4-2: Архитектурные модели содержат сведения об использовании.

Подобные структурные модели могут отображать физическую архитектуру, как показано на примерах выше, но могут использоваться и для отображения логической структуры. Логические структуры все чаще используется при разработке сложных систем, поскольку они позволяют инженерам обосновывать функциональные взаимодействия элементов без создания конкретной физической структуры. Вернемся к автомобилю на рис. 4-1. Еще несколько десятилетий назад автомобильные подсистемы (тормоз, двигатель, радио) были совершенно обособлены, и для их понимания и создания было вполне достаточно иметь только их физическую структуру. Сейчас же, если мы рассмотрим логическую структуру автомобиля, мы обнаружим следующие элементы:

- Силовая установка, которая уже вторглась в прежние физические границы тормоза, электрооборудования, двигателя, системы управления
- Управление информационными потоками, к которому относится вся информация, - полученная, сохраненная, отправленная и используемая - внутри автомобиля
- Система развлечений, которая, в конечном итоге, может иметь общие компоненты как с силовой установкой, так и с системой управления информацией

- Система безопасности, которая может также быть сопряжена со многими другими системами автомобиля, например, системой торможения, системой управления информацией, интерфейсом водителя

Рассматривая логическую структуру отдельно, и, в идеальном случае, до ее физического воплощения, системные инженеры могут найти оптимальные, более изящные решения. Двойственный - логический\физический - архитектурный подход — это единственно верный путь к контролю и управлению сложностью новых, интегрированных умных продуктов, которые создаются уже далеко не так, как дедушкина старая Волга.

Моделирование поведения системы

Для понимания поведения системы вам необходимо понимать, как взаимодействуют физические и логические компоненты архитектуры системы. Модели поведения системы призваны отображать динамическую информацию о системе: переход из одного состояния в другое, ответная реакция системы на определенные события, обмен данными между взаимодействующими компонентами. Помимо этого, такие модели иллюстрируют потоки данных и команд управления между некоторыми активностями, например, как выходные данные от Датчика ступицы поступают на Антиблокировочный блок управления или как команды управления передаются из одной части системы в другую.

Рис. 4-3 содержит упрощенную диаграму состояний, которая отображает операционные состояния автомобиля. Машина пребывает в одном из четырех состояний - выключено, режим ожидания, ускорение или торможение, — до тех пор, пока не произойдет определенное событие (например, нажатие тормозной педали), которое заставляет машину перейти из одного состояния в другое.

Вы можете составить схему связей между моделями поведения и структурными моделями, используя процесс *локализации*, чтобы привнести связность и логичность. Например, действие «определить потерю сцепления» локализуется в компоненте «датчик потери сцепления» в вашей структурной модели системы, а «регулировать силу тормозного усилия» локализуется в компоненте «регулятор тормозного усилия».

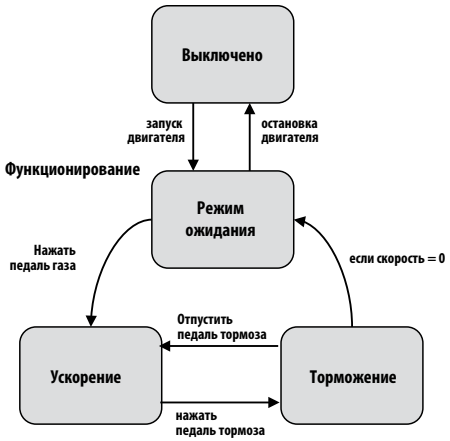

Рис. 4-3: Поведенческая модель, отражающая переходы из одного рабочего состояния в другое.

Для сложных систем вам потребуется собрать в единое целое массу маленьких моделей поведения, чтобы отобразить все активности, которые происходят в системе. Во многих случаях одна активность запускает следующую в другой части системы, второе событие запускает третье и т.д. - таким образом индивидуальные модели оказываются соединенными друг с другом. На момент завершения построения моделей всех активностей перед вами предстанет большая, сложная, однородная и последовательная модель поведения всей системы.

Использование стандартных конструкций и семантики моделирования, например, языка моделирования систем (SysML), позволяет затем использовать коммерческие инструменты, которые могут автоматизировать исполнение моделей поведения системы. Эти инструменты автоматически «переводят» существующие конструкции моделей (например, диаграммы переходов) в наборы операторов «*если - тогда - выполнить*». Этим способом можно имитировать поведение системы на уровне программного обеспечения, проанализировать варианты «*что-если*», рассмотреть альтернативные варианты дизайна и провести анализ влияния еще до создания системы как таковой. И если на исследование затрат обычно тратятся часы, а то и дни, то в данном случае счет может идти на минуты.

Построение моделей

Отраслевые стандарты (такие, как SysML) являются основой для общего понимания системного моделирования на всех его стадиях. Модели SysML состоят из набора *диаграмм*, которые отображают

структуру, поведение, требования и количественные ограничения системы. Диаграммы SysML «покрывают» четыре различных области:

- **Структура:** Описывается какие архитектурные элементы (логические и физические), которые также известны как *блоки*, содержатся в системе или подсистеме, а также их взаимосвязи. Вот, например, подсистема Антиблокировочного блока управления содержит Датчик силы сцепления и Регулятор тормозного усилия.
- **Поведение:** Описывается как ведет себя система или подсистема, включая смену состояний, последовательности активностей, функции и взаимодействия. (Например, "*определить потерю сцепления*" запускает "*изменение тормозного усилия*").
- **Требования:** Описываются конкретные требования к системе или подсистеме. (Например, требования к тормозному пути).
- **Параметрия:** Описываются ограничения параметров, накладываемых на систему или подсистему. (Например, ограничения к автомобилю, который предполагается эксплуатировать в особых зимних условиях).

Четыре стадии моделирования системы

Сами системы по своей природе являются рекурсивными структурами, состоящими из множества уровней - на самом высоком уровне находится система как таковая, которая далее декомпозируется на подсистемы, а они, в свою очередь, — на компоненты. Поэтому самый лучший способ для моделирования системы – это использовать многошаговый рекурсивный процесс, который начинается на высшем уровне и затем проходит четыре стадии. Эти стадии помогают максимально упростить построение модели и управлять сложностью:

- **Контекст:** Здесь вы устанавливаете границы вашей системы, определяете людей и другие системы, с которыми ваша система взаимодействует (известные также как *действующие лица*), и описываете интерфейсы (как у них происходит обмен информацией с системой и какими данными они обмениваются). Все вместе эти элементы контекстной модели описывают систему и ее непосредственное окружение. Для этих целей используются блок-диаграммы SysML.
- **Использование:** Здесь вы описываете все способы использования системы действующими лицами, включая описание того, кто и как использует систему, а также кого и как использует система. Это лучше всего делать в виде конкретных пошаговых повествовательных описаний использования системы, используя прецеденты использования и иллюстрируя диаграммами активностей SysML. Здесь следует максимально детализировать системные требования, конкретизировать их и придать им законченный вид (помните, что в исходных требованиях может

отсутствовать очень много важных подробностей). Позже эти же сценарии использования станут основой для тестирования системы – тогда вы будете базироваться на тех ваших знаниях, что получили сейчас из анализа использования системы.

- **Реализация:** Здесь вы определяете структуру (архитектуру) и поведение (функционирование) моделей, которые вместе описывают как каждый сценарий использования реализуется системой за счет взаимодействия элементов внутри архитектуры системы. Требуемое поведение реализуется (обретает реальный вид) в элементах системы. Надо сказать, что этот этап существенно отличается от традиционных методов разработки систем, подразумевающих вначале формирование требований для конкретных физических компонентов, а затем упование на то, что в результате разработки система заработает, как надо. Здесь как раз дается четкое и недвусмысленное описание использования, после чего вы создаете систему на базе этих конкретных сценариев, что дает вам *уверенность* в том, что система создается, чтобы соответствовать заданным требованиям по ее использованию.
- **Исполнение:** Здесь вы исполняете модели поведения, чтобы продемонстрировать, что ваш дизайн соответствует требованиям. Простые исполняемые модели даже при высокой степени абстракции — это эффективный и экономичный способ как можно раньше вскрыть сложные проблемы, устранить непонимание или неясность, обнаружить пропущенные или двусмысленные требования, а также другие неприятные моменты, способные отрицательно повлиять на график и объем работ. Все это позволяет всем участникам процесса думать и действовать одинаково правильно еще до того, как что-то началось реально воплощаться.

Начинать следует с самого верхнего уровня декомпозиции вашего дизайна - с уровня системы (тот же Микроавтобус). После определения контекста и построения моделей использования, исходя из системных требований, создаются высокоуровневые архитектурные и поведенческие модели. Затем эти модели исполняются для того, чтобы продемонстрировать, что они действительно делают то, для чего предназначена система. После завершения этого процесса по отношению к системному уровню, следует повторить его для следующего уровня декомпозиции — уровня подсистем.

Продолжая пробираться сквозь уровни декомпозиции, уточняя и меняя контекст по мере обращения к модели каждого уровня, вы достигаете самого нижнего уровня – уровня компонентов, где вы конкретизируете физическое воплощение вашего дизайна (например, проектирование электроники, программного обеспечения, механического оборудования).

На каждом уровне, перемещаясь по горизонтальным уровням V-модели, следует пользоваться процедурами валидации и верифика-

ции системы, используя модель как базис. Используйте исполняемые модели, а также все возможные математические и другие доступные симуляторы, чтобы как можно чаще верифицировать систему, прежде чем вы приступите к фазе ее реализации и внедрения.

Ваша заветная цель — это построение полноценной модели системы, точнее — полной виртуальной версии реальной системы. Конечно, этот результат пока не достижим полностью, но инструменты моделирования постоянно совершенствуются, они учатся взаимодействовать в разных инженерных дисциплинах, так что победа близка.

Почему моделирование это не просто дань моде?

Может создаться впечатление, что моделирование архитектуры и поведения сложной системы приносит больше хлопот, чем оно того стоит, — но это только до тех пор, пока вы не вспомните о проблемах вашего прошлого проекта, когда никто из команды разработчиков не мог признать, что упустил из виду одно из критических требований, и когда потом сама испанская инквизия казалась веселой вечеринкой по сравнению с последующим «разбором полетов на ковре у начальства» и обсуждением загубленного проекта.

Моделирование позволяет вам четко представить все проблемные детали дизайна системы и отобразить ее в виде упорядоченной структуры, позволяющей вам визуализировать, понимать и усваивать всю сложность архитектуры и поведения системы. Моделирование позволит вам проверить различные варианты архитектуры и технических решений системы, проводить исследование уровня затрат, оценивать влияние изменений до того, как вы начали реально воплощать в жизнь ваши задумки, - тем самым снижая риски и издержки, связанные с разработкой.

Модели обеспечивают контекст, который служит основой для обсуждения системы на различных уровнях. Используя модели, вы получаете возможность исследовать самые разные аспекты разработки системы: планирование, требования, архитектуру, реализацию, развертывание, поведение системы, входные и выходные данные, и т.д. — таким образом вы и ваши коллеги можете с легкостью обосновывать как всю систему в целом, так и ее конкретные детали.

Использование языков и техник, признанных отраслевыми стандартами моделирования, таких, как SysML, устраняют неточности и неоднозначности, помогают преодолеть языковые барьеры в многонациональных коллективах разработчиков, а также дают единственную исходную точку для определения статуса проекта и документирования. Улучшение взаимодействия и четкая и точная документация повышают продуктивность, сокращают время разработки, а самое главное — обеспечивают высокий уровень качества.

Глава 5

Обеспечение высшего уровня качества

В этой главе

- Качество как неотъемлемый элемент процесса разработки систем
- Циклическое тестирование и проектирование
- Использование моделей для раннего выявления ошибок

Раньше считалось, что обеспечение качества есть такой процесс, который проводится только в самом конце цикла разработки, как если бы качество было некой материальной деталью, которую вы могли «прикрутить» к продукту перед его поставкой. Но при этом, когда обнаруживался дефект, требовались невероятные усилия, чтобы найти причину неполадок и устранить проблему.

В современном мире грандиозных по сложности и ведомых программным обеспечением интеллектуальных продуктов качество должно стать неотъемлемой частью процесса разработки систем, чтобы обеспечить не только бесперебойную работу полученного продукта, не имеющего дефектов, но и обеспечить его полное соответствие своему предназначению. В это главе вы познакомитесь с тем, как системная инженерия посредством валидации и верификации помогает справляться с проблемами качества на самых ранних цикла разработки, повышая шансы успешного завершения проекта.

Уровни тестирования

Когда вы создаете интеллектуальный продукт, который состоит из нескольких подсистем, содержащих тысячи (а то и миллионы) строк кода, вам, волей неволей, приходится задумываться над тем, как же обеспечить верификацию и валидацию всей системы. С чего стоит начать? Как убедиться в том, что элементы мозаики не только *сложились* в правильную картинку, но и *работают* правильным образом, чтобы обеспечить такую функциональность продукта, которой интересовались ваши заказчики? Ответ прост - обеспечение качества стоит начинать с самых первых шагов проектирования системы.

Крупные, сложные системы декомпозируются на несколько уровней, включая уровень системы, один или несколько уровней подсистем и, наконец, уровень компонентов.

В то время как кажется вполне логичным тестировать систему после того, как вся она собрана в единое целое, тем не менее передовые практики системной инженерии включают в себя более сложные техники обеспечения качества по ходу создания системы. По мере создания каждого компонента системы (будь то аппаратное или программное обеспечение), он вначале тестируется и только лишь затем интегрируется с другими компонентами, чтобы сформировать общую подсистему. Затем тестируется подсистема, которая интегрируется вместе с другими подсистемами, чтобы сформировать систему, которая затем также тестируется. И, наконец, происходит тестирование самой системы в ее операционном окружении (контексте), чтобы убедиться, что система выполняет свое предназначение в реальных условиях.

Помимо тестирования системы по мере ее создания, можно проводить процедуры верификации и валидации на моделях и симуляторах системы, чтобы несколько ранее выявлять некоторые проблемы - еще до того, как вы начнете воплощать что-то в металле или создавать первую микросхему. Верификацию можно проводить еще на начальном этапе сбора требований, чтобы убедиться, что ваше понимание и интерпретация потребностей заказчика действительно отвечают его нуждам.

Тестирование компонентов

Тестирование на самых низких уровнях системы тесно связано с конечной реализацией вашего дизайна. Вы тестируете каждый аппаратный или программный компонент, а при обнаружении дефектов вносите изменения в первичные задумки. Если это касается программного обеспечения, то это может повлечь за собой несколько итеративных циклов написания кода, его тестирования, внесения изменений в код – и так до полной его отладки.

После того, как вы устранили неполадки, следует проверить, что компонент соответствует предъявляемым ему требованиям. Помните этап разработки подробного дизайна вашей системы? Именно тогда частью фазы детальной проработки дизайна являлось не только формирование требований уровня компонентов, но и составление *Плана верификации компонентов*. Вот и пришла пора использовать сценарии тестирования из этого плана для проверки соответствия компонента требованиям. Если во время тестирования вы обнаружили дефекты, то необходимо вернуться на этап реализации компонента и внести соответствующие коррективы. Только после тщательного исследования и проверки всех компонентов, они могут считаться готовыми к интегрированию в более высокоуровневые узлы или подсистемы.

Убедитесь в том, что План верификации компонентов содержит четкие задокументированные сценарии тестирования и ожидаемые результаты для каждого компонента, а также в том, что у вас имеется и вы используете матрицу трассируемости, которая связывает эти тесты и исходные требования. Такой подход гарантирует, что если все тесты пройдены успешно, то ваша система соответствует всем предъявляемым требованиям.

Интеграция и верификация подсистем

Полностью верифицированные компоненты аппаратного и программного обеспечения готовы к интеграции в модули или подсистемы. А если вы уже проводили предварительное тестирование интерфейсов, то этот этап должен пройти достаточно гладко.

Цель этой стадии тестирования – это убедиться в том, что все интерфейсы между узлами и компонентами были правильно реализованы, а также в том, что все требования и ограничения к подсистеме были соблюдены.

В зависимости от сложности вашей подсистемы может возникнуть необходимость в разработке *Плана интеграции*, который будет определять порядок интегрирования низкоуровневых компонентов и подузлов. Имеет смысл так планировать интеграцию, чтобы те элементы подсистемы, которые должны работать вместе, интегрировались, по возможности, в одно и то же время. На каждом шаге интеграции верифицируйте функциональность получаемого набора элементов по отношению к соответствующему набору требований, используя *План верификации подсистем*, который вы также создавали на этапе разработки дизайна.

Будьте внимательны, чтобы не упустить из виду результаты тестов, которые вы выполняли для верификации требований уровня компонентов, - ведь многие из этих требований при декомпозиции системы фактически пришли (посредством каскадирования) с самого верхнего уровня системы. Например, если системные требования требуют, чтобы экран дисплея был чист после нажатия кнопки пользователем, вначале необходимо проверить способность самого компонента, отвечающего за дисплей, очищать экран (тест на уровне компонента), а затем проверить срабатывание этой функции по нажатию кнопки (тест на уровне подсистемы).

Тестирование системы

Проходя через серию прогрессирующих итераций, вы интегрируете, тестируете и верифицируете подсистемы до тех пор, пока не достигните уровня системы (см. рис. 5-1). Каждая итерация подразумевает тщательное и всестороннее тестирование (с особым акцентом на интерфейсы) и верификацию, подтверждающую реализацию соответствующих требований. На самом высоком уровне вы выполняете

тестирование, чтобы убедиться, что вся система отвечает высокоуровневым системным требованиям, сформированным ранее в процессе проектирования. Повторимся - при проведении испытаний следует придерживаться плана верификации, созданного одновременно с формированием системных требований.

Очень важно фиксировать результаты каждого теста и отмечать любые неожиданные отклики, отклонения или другие аномалии. Однако следует сопротивляться искушению сразу же бросаться исправлять обнаруженные дефекты – в этом случае вы рискуете потерять контроль над конфигурацией. Будет правильней задокументировать проблему, проанализировать причины ее возникновения и определить план дальнейших действий по ее устранению в рамках систематического процесса.

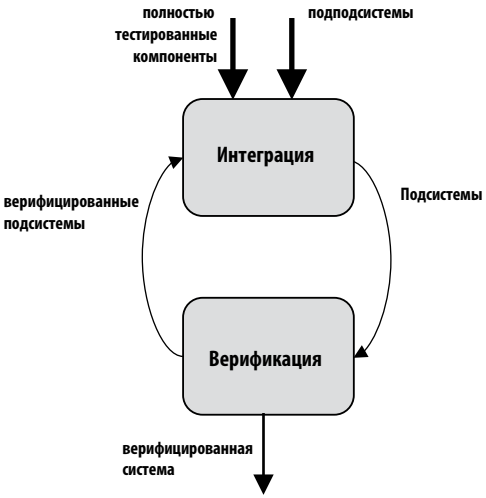

Рис. 5-1: Интеграция и верификация это итерационный процесс.

При благоприятном развитии ситуации вы получаете верифицированную систему, которую можете демонстрировать заказчикам. Вам останется только показать, что все системные требования удовлетворены, что доказывает правильность реализации системы.

Приемочные испытания системы

«Был бы дом, а жители найдутся!» - Если вы исповедуете эту философию, вам далеко не уйти. Ваша система может быть идеально спроектирована и реализована, может удовлетворять всем требованиям и прочим задумкам, но, если она не соответствует своему предполагаемому предназначению, то она обречена на неудачу.

Возьмем, к примеру, хорошо продуманную автоматическую систему управления светофорами. Мастерски реализованная система, предназначенная для управления очередностью переключений сигналов светофоров в большом городе, она не будет соответствовать своей *намеченной* цели — снижению количества дорожных пробок за счет оптимизации потоков движения, — если никто не удосужиться предварительно изучить типичные направления движения в городе и не спроецировать их на системные требования, чтобы потом учесть при формировании временных интервалов и очередности переключений светофоров.

Цель приемочных испытаний системы состоит в том, чтобы подтвердить, что система соответствует своему предназначению.

В течение концептуальной стадии вашего проекта вы определяли ключевые потребности заказчика, общие технические характеристики системы, сценарии применения (концепции эксплуатации и прецеденты использования), а также устанавливали критерии эффективности для последующей валидации системы. Вы также составили *План валидации системы*, который, если сообразили, засунули в сейф и не позволяли вносить в него изменения, чтоб никому не пришло в голову менять его с целью сэкономить пару тысяч рублей. План валидации системы – это непоколебимая скала, это основа, которая поможет вам доказать, что система соответствует целям ее создания.

Далее следует провести валидацию системы в работе с реальными пользователями, оценить плановые показатели эффективности ее работы, возможно и с точки зрения удовлетворения нужд заказчика. Валидация может потребовать сбора данных до, в процессе и после развертывания системы. Зафиксировав и задокументировав все показатели производительности системы, можно смело встречаться с заказчиком, совместно анализировать полученные данные и определять успешность проекта.

Если проблемы в системе обнаруживаются на этапе тестирования, то это является дефектом системы, который еще возможно исправить, внеся изменения в саму систему. Однако следует заметить, что проблема может оказаться и в другом месте. Если ваш план верификации или тестовая процедура некорректны или устарели, то тест может не пройти отнюдь не потому, что в самой системе что-то не так.

Аналогично, если требования ошибочны, неоднозначны, или неполны (особенно это касается интерфейсов), то это также может быть источником проблемы - тем больше оснований со всей тщательностью подходить к формированию требований и программы испытаний в самом начале «игры».

Если в процессе верификации обнаружилась проблема, следует вернуться в начало процесса и проверить ваши требования и дизайн, внести необходимые изменения и повторить процедуру проверки.

Представим себе ситуацию, когда вы проектируете подсистему автомобильных стеклоочистителей (больше известных как *дворники* или *щетки*), оборудованных датчиком дождя. Цель подсистемы - запуск стеклоочистителей при обнаружении капель дождя на внешней поверхности ветрового стекла. Высокоуровневая архитектура вашей подсистемы содержит оптический датчик (с определенным диапазоном срабатывания), который крепится к внутренней поверхности ветрового стекла, электронное управляющее устройство и программное обеспечение. Ваш дизайн таков, что заставляет взаимодействовать оптический датчик и программное обеспечение, чтобы проверить наличие воды на стекле, и, при необходимости, привести в действие стеклоочистители.

Вы тестируете отдельные компоненты и подтверждаете, что подсистема работает корректно в рамках диапазона срабатывания датчика. Затем стеклоочистители, ветровое стекло и другие подсистемы интегрируются в автомобиль. И вот при тестировании подсистемы уже в рамках всей системы – подумать только!! – она не работает.

После интенсивного анализа и поиска причин вы обнаруживаете источник проблем - физические характеристики ветрового стекла (в частности, оптические показатели и толщина) несовместимы с рабочим диапазоном оптического датчика. И тут вы осознаете, что в свое время вы не сформировали такие требования к физическим характеристикам ветрового стекла, которые и должны были обеспечить совместимость. В этом случае вам следует вернуться на стадию проектирования, добавить требования и изменить дизайн ветрового стекла.

Как видите, причиной переделок послужило ничем не подтвержденное *допущение*, – страшный сон системных инженеров. В данном случае допущение состояло в том, что кто-то предположил, что оптический датчик будет работать с любым материалом (типом) ветрового стекла. Разумеется, если бы такая мысль была высказана вслух, это несоответствие сразу бы бросилось в глаза и кто-то обязательно бы возразил, а затем уж были бы сформированы пропущенные требования. Этот болезненный опыт является хорошим примером того, почему так важно тестировать часто и на самых ранних этапах.

Тестировать раньше, тестировать чаще

Чем раньше вы обнаружите дефект (ошибку, нарушение, недостаток, неисправность и т.д.), тем дешевле вам обойдется его исправление. Как показано на рис. 5-2, затраты на исправление дефектов, обнаруженных после выпуска продукта, почти в 100 раз превышают издержки на исправление, если недостатки обнаружились в процессе формирования требований. И эта диаграмма только один из многочисленных примеров. Четко следуя «безжалостному» процессу тестирования, вы можете радикально снизить затраты на устранение недоработок.

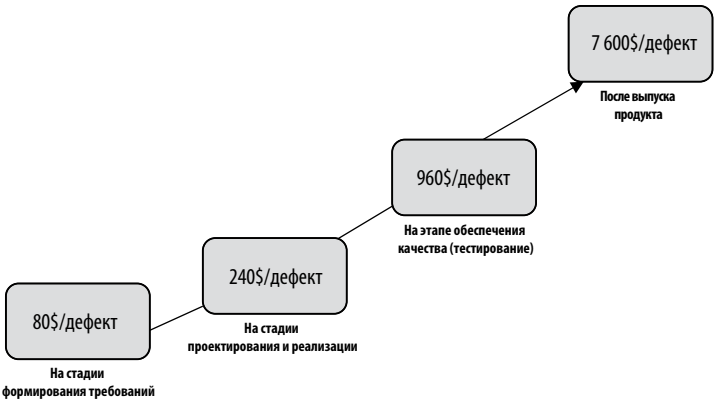

Рис. 5-2: Стоимость исправления дефектов значительно возрастает в процессе разработки.

Дьявол кроется в ... интерфейсах

Одна из самых больших проблем в разработке и тестировании крупных многокомпонентных систем заключается в том, что огромное количество независимо разрабатываемых блоков, будучи интегрированными в общую систему, не всегда работают вместе так, как было задумано. Если же, *прежде* чем приступить к процедуре интегрирования и тестирования, вы предполагаете ждать, когда все блоки будут разработаны и готовы, а потом обнаружите проблемы интеграции, то скорее всего вы уже в весьма затруднительном положении - все сроки будут сорваны и остаться в рамках бюджета вам не удастся.

Именно этот фактор - обнаружение проблем интеграции на поздних этапах разработки - возможно является самым основным фактором, который приводит к выпадению из графика и перерасходу средств. Это очень *большой* фактор риска на пути к успешному завершению программы.

Что же тут можно сделать? Системные инженеры придумали два основных подхода для минимизации этого риска:

- Следует тестировать интерфейсы и проверять взаимодействие между ключевыми подсистемами и компонентами *на ранних этапах*, используя модели и симуляторы вместо некоторых (или даже всех) подсистем и компонентов.
- Следует постепенно (или циклично) интегрировать части системы и проверять их работу, и при этом всячески избегать ситуации, при которой вы ждете готовность всех узлов и только потом начинаете интеграцию, - и поиск неизбежных проблем.

Первый способ — моделирование систем (см. Главу 4) — может оказаться очень полезным. Модели использования, выраженные в диаграммах активностей и диаграммах последовательности, помогут вам

определить какие узлы системы общаются между собой и в каких интерфейсах они нуждаются. Имея эти знания, вы сможете намного раньше попытаться интегрировать между собой именно те узлы системы, что общаются между собой, что позволит снизить число симуляторов и имитаторов. Если вы сможете заставить взаимодействующие команды, которые разрабатывают разные подсистемы, начать работать вместе как можно раньше, то это сдвинет обнаружение проблем на более ранние этапы разработки, когда эти проблемы легко и дешевле исправить.

Модели также позволяют вам проверить правильно ли вы понимаете взаимодействие между подсистемами, — исполняя модели легко убедиться, что они ведут себя так, как вы себе это представляете. Транслируя идею — через диаграмму — к модели и ее исполнению, можно обнаружить несоответствие в поведении в то время, когда исправить проблему будет намного проще и дешевле.

С помощью моделей, которые имитируют работу еще не созданных компонентов, можно тестировать интерфейсы задолго до того, как вся система обретет плоть и кровь. Таким примером может служить симулятор самолета, который дает возможность инженерам тестировать многие аспекты поведения самолета еще до того, как реальный аппарат оторвется от земли.

Второй способ — это внедрить в жизненный цикл разработки как можно больше последовательных интеграций и тестирований (итераций). Системные инженеры уже много десятков лет пользуются итеративным жизненым циклом разработки. Разумеется, при разработке программного обеспечения использовать итеративный подход куда проще, чем, скажем, при создании электронной аппаратуры или корпуса летательного аппарата (хотя бы потому, что создание оборудования требует больше времени), но определенные принципы можно применить и здесь.

Интегрируя комбинацию прототипов или симуляторов аппаратного обеспечения с новыми разработками программного обеспечения, ранние итерационные циклы можно нацелить на те компоненты продукта, которые имеют высокую степень риска. С такими предварительными, но при этом рабочими сборками системы, вы можете решить проблемы интеграции, совершенствовать интерфейсы и контролировать затраты.

Некоторые наиболее дальновидные системные инженеры предлагают даже вначале тестировать интеграцию и только потом тестировать отдельные компоненты, как часть интегрированной подсистемы. На первый взгляд, этот подход покажется парадоксальным и противоречащим здравому смыслу, но фактически это является ранней профилактикой проблем интеграции, поскольку вам не приходится дожидаться полной интеграции системы.

Глава 6

Давая большим коллективам возможность взаимодействовать и управлять изменениями

В этой главе

- Поиск общей позиции в целях обеспечения эффективного общения и взаимодействия
- Автоматизация рабочих процессов
- Сопрягая инструменты с данными жизненного цикла

В небольших командах разработчиков вопрос эффективности совместной работы не стоит: они умеют делиться информацией, применяют одинаковые инструменты разработки, сообщают друг другу о вносимых в требования изменениях, вместе ищут дефекты и легко договариваются о встрече в баре в пятницу.

А вот теперь увеличьте число работников в команде, скажем, раз в 100, вручите им список из 700 требований, объявите, что у них есть всего полгода на разработку продукта — и можете прощаться со своей работой (и про вечеринки тоже можно забыть).

Как же сохранить секрет взаимодействия маленьких команд и наладить такое взаимодействие в крупных коллективах? Решением этой задачи мы и займемся в этой главе.

Учимся у Facebook

Неимоверно успешная история развития социальной сети Facebook может стать источником самых разных хороших идей. Представленный в 2004, когда большинство пользователей уже пользовались электронной почтой, Facebook мгновенно набрал обороты, и к началу 2011 г. насчитывал уже 600 миллионов участников. Что же сделало его таким популярным?

Основатели Facebook нашли способ создать дружественную платформу реального времени для социального общения, используя структуру и взаимосвязи, которые окружают нас в жизни: друзья, родственники, единомышленники. Так появился дружественный интерфейс обмена новостями, фотографиями, мнениями, лайками (like), статусами отношений и другой информацией. Вместо того, чтобы заставлять людей общаться в формате, типичном для информационных технологий, как делает это электронная почта, Facebook поставил технологи на службу человеческому общению.

Теперь давайте представим сдвиг «парадигмы Facebook» в сторону системной инженерии. Представим себе некую технологическую систему, которая использует для своей выгоды способ, с помощью которого взаимодействуют между собой разработчики, инженеры, проектировщики и другие заинтересованные лица. Подобно Facebook такая платформа будет опираться на интернет, как на средство объединения и сближения людей, создавая тем самым огромный – хоть и рассредоточенный в реальности - коллектив, не уступающий по эффективности команде, которая работает в одном помещении.

Синхронизация мышления

Чтобы создать востребованный на рынке интеллектуальный продукт, требуется чертовски много еще чего помимо просто гениальной инженерии. Как показывают опросы, примерно треть всех произведенных в мире устройств не соответствуют требованиям по функциональности или производительности, при этом 24% всех проектов закрываются с непоправимым отставанием от графика. И очень часто причина катастрофической неудачи системы не имеет отношения к системной инженерии. Чаще всего проблемы вызваны недостатком сведений, знаний или слабой коммуникацией.

Неудивительно, что разработка крупных систем в первую очередь страдает от слабого взаимодействия. Почти всегда крупный проект – это множество городов, компаний и даже стран. Язык и культурные различия – серьезное препятствие для общения, которое усугубляется разницей во времени между часовыми поясами. Даже работникам одной компании отсутствие четкой организационной структуры может мешать нормальному общению, снижать производительность и способствовать перекладыванию проблем “с больной головы на здоровую”.

Недостаток информационного обмена может привести ко многим проблемам, среди которых:

- Отсутствие четких системных целей
- Множественные и различающиеся интерпретации системных требований
- Неполный список требований любого уровня

- Затраты времени на поиск и систематизацию информации «вручную» из многочисленных источников
- Работа команд с устаревшими документами
- Дублирование или нехватка ответственности исполнителей

Добавьте сюда еще постоянное давление начальства на команды разработчиков по поводу необходимости повышения эффективности, даже если сложность системы все возрастает. При этом следует учитывать, что умные продукты со встроенным программным обеспечением требуют больших объемов документации. Да и кривая нагрузки, связанная с обучением новых членов команды, весьма крутая.

Наилучший путь преодоления трудностей коммуникации, присущих крупным коллективам, - это создание общей платформы разработки и поддержки, а также введение единого языка коммуникации, используемого в рамках этой платформы.

Закладка фундамента

Большинство сегодняшних интеллектуальных систем содержат множество подсистем, поставляемых из различных источников, и насчитывают миллионы строк кода – и все это создано инженерными командами из разных компаний, стран и культур. Возьмем современный самолет: его корпус производится в одной стране, двигатель — в другой, бортовая радиоэлектроника — в третьей, а встроенное программное обеспечение — в четвертой. Чтобы процессы разработки и тестирования протекали гладко, очень важно использовать единую платформу для разработки систем.

Использование общей платформы разработки разрушает барьеры между разными командами, позволяя инженерам работать вместе на протяжении всего жизненного цикла. Унифицированная платформа облегчает распределенным командам интеграцию их наработок и обмен информацией, – и это существенно экономит время. Слаженная работа специалистов и упорядоченный производственный процесс существенно повышают качество разработок и способствуют тому, что коллектив получает удовольствие от работы. Более того, инженерам становится проще информировать любого члена команды о текущем статусе проекта, используя панели мониторинга, что облегчает работу менеджеров и помогает им держать проект в рамках установленного срока и бюджета.

Говорим на одном языке

Чтобы объединить работников разных команд, исповедующих разные культуры и работающих с разными инженерными дисциплинами, нет ничего более эффективного, чем использование подхода на

основе моделей, который базируется на общем языке, не связанном с конкретной предметной областью. Обеспечивая визуальное отображение системного дизайна, моделирование устраняет языковые барьеры, облегчает понимание системы каждым членом команды, ускоряет обмен междисциплинарными знаниями. Общее понимание конвертируется в улучшение продуктивности, поскольку инженеры больше не тратят время на исправление недоразумений и переделки дизайна.

В сфере разработки систем на основе моделей становится стандартом язык системного моделирования (SysML). Он поддерживает все фазы системных разработок, включая спецификации требований, системный анализ и дизайн, верификацию и валидацию систем, которые строятся из аппаратного и программного обеспечения, данных, людей и даже средств обслуживания. Помимо этого, SysML абсолютно совместим с унифицированным языком моделирования (UML), что существенно упрощает процедуру трансляции моделей с системной перспективы в перспективу программную.

Существует множество доступных коммерческих приложений, поддерживающих разработку на языке SysML, и каждое использует свою собственную уникальную рабочую среду. Чтобы повысить эффективность взаимодействия, командам лучше стандартизировать общую рабочую платформу, которая базировалась бы на наиболее эффективных инструментах, доступных на рынке.

Больше, чем совместное использование почты и документов

Создание общей платформы разработки — это уже гигантский шаг в организации командной работы и развитии эффективного взаимодействия, но это еще далеко не все. Если один участник команды внес изменение (например, в требование или в модель системы), и эта информация не становится тут же доступной остальным участникам команды, это ведет к хаосу и необратимым последствиям.

Отслеживание изменений

Когда вы создаете систему, критически важная информация постоянно меняется – из-за дизайна. Системная инженерия подразумевает (помимо прочего) итеративные процессы проектирования и тестирования: вы проектируете вашу систему, разрабатываете модели, тестируете модели, потом, если обнаружили дефект, вносите изменения в систему и так далее. В общем, вы постоянно обновляете модели, результаты тестирования и прочую информацию. Требования также могут меняться, - по мере изменения ситуации на рынке и потребностей бизнеса.

Традиционно крупные инженерные компании ориентируются на текстовую документацию, как на источник всей информации, связанной с проектом. Полки просто забиты папками со списками требований, с документами по высокоуровневой архитектуре, с проектной документацией, а также любыми другими документами, составляющими базу знаний по системе и ее окружению. Но такого рода документы имеют существенное ограничение – они лишь *записывают и хранят* информацию, а вот с изменениями информации не так все просто. Если вы надеетесь и опираетесь только на документацию, то может оказаться, что она устареет еще до того, как будет утверждена.

Команды, разрабатывающие сложные системы, нуждаются в согласованном и гибком механизме для сбора и хранения критической информации, позволяющем легко обновлять данные, не нарушая их целостности.

Обмениваясь информацией об изменениях

Традиционный метод, когда вы создаете десятки важных документов и рассылаете их по электронной почте, в современном мире просто не может работать хорошо. Когда только количество требований к сложным системам исчисляется сотнями, а то и тысячами единиц, надеяться только на электронную почту значит семимильными шагами двигаться в сторону неразберихи и хаоса.

Команды, занимающиеся системными разработками, нуждаются в средстве, способствующем эффективному и оперативному обмену информацией между всеми участниками проекта. Прочная и надежная платформа обмена информацией - это единственный способ избавиться от проблем, которые связаны с тем, что в работе может циркулировать целый ворох разных версий важных документов.

Какой информацией стоит обмениваться ?

Если вы попытаетесь составить список всей критически важной информации, которую необходимо контролировать, чтобы создать сложную систему, то вполне возможно, что ваша попытка будет обречена на неудачу. Поэтому, когда вы задумываетесь над тем, какой информацией следует обмениваться участникам вашей команды, не следует обрушивать на каждого участника проекта все доступные данные, описывающие систему.

Ваша цель состоит в том, чтобы обеспечить обмен информацией и взаимодействие участников команды, но не перегружать их лишними подробностями. Так что стоит составить перечень той минимальной информации, которая необходима для обмена, чтобы улучшить взаимодействие участников, например такой:

- Приоритеты разработки
- Утвержденные решения по проекту
- Графики выполнения работ
- Роли участников команды и сферы их ответственности
- Требования
- Запросы на изменения
- Концептуальные модели
- Прецеденты использования
- Планы проведения тестов и испытаний
- Обнаруженные дефекты
- Критические проблемы
- Данные о трассируемости
- Сведения о бюджете
- Сведения о закупках

Какие способы упрощают обмен информацией

Разумеется не совсем реально ожидать от каждой из нескольких сотен компаний, участвующих в процессе создания продукта, использования одного и того же набора инструментов от одной компании-вендора. А значит должен существовать способ, которым каждый участник этого виртуального коллектива может обмениваться данными разработки, - независимо от того какой командой или партнером они созданы.

Эффективное взаимодействие начинается с единой платформы, с помощью которой весь процесс разработки может контролироваться и управляться. Используя эту платформу, все заинтересованные и вовлеченные стороны будут создавать и обрабатывать данные, которыми они будут обмениваться, анализировать их, отчитываться об эффективности и экономичности всего цикла разработки.

Автоматизированные и интегрированные инструменты, которые поддерживают технологии беспрепятственной коммуникации и автоматизируют производственный процесс – это следующий важный уровень организации взаимодействия.

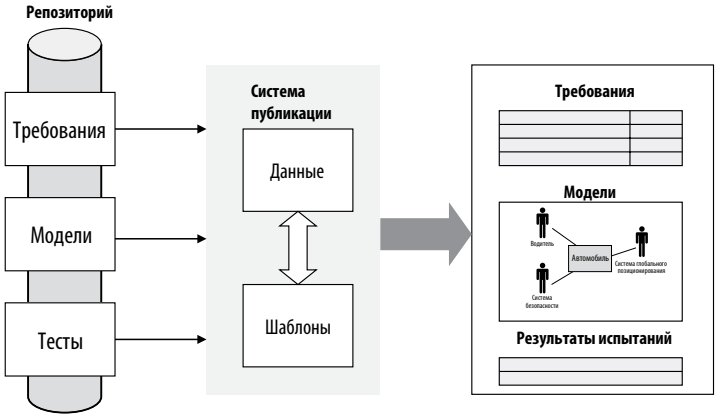

Рис. 6-1: Виртуальный репозиторий поможет улучшить взаимодействие.

Создание виртуального репозитория оперативной информации

Всем участникам проекта необходим постоянный в режиме on-line доступ к актуальной проектной информации – независимо от места расположения их команды и формата требуемых данных. Обычно системные инженеры резервируют в графике работ достаточно большое количество времени на обмен информацией и последующее подтверждение того, что каждый, кто вовлечен в работу, пользуется одними и теми же версиями документов.

Систематизируя критически важную информацию в виртуальном репозитории, вы создаете единый источник знаний по основным аспектам процесса разработки и можете автоматизировать процесс обмена информацией.

Виртуальная база данных не обязательно должна существовать физически как единое целое (отсюда и ее название). Более того, по мере необходимости требуемые данные даже удобнее извлекать из глобально распределенных источников. Например, данные по механической инженерии могут извлекаться из репозитория, который поддерживается командой инженеров-механиков и расположен в Сиэтле, в то время как данные по электрической инженерии можно получить из соответствующей базы инженеров-электриков в Токио. Любому члену команды совсем необязательно знать (тем более – заботиться) о том, где физически базируются необходимые данные — главное, что они доступны тогда и там, когда и где необходимы.

На рис. 6-1 представлена концептуальная модель такого виртуального репозитория и вариант ее использования. Менеджер бизнес-процессов

играет роль интеллектуального центра, отвечающего за надлежащее исполнение всех активностей, связанных с разработкой. Использование открытых стандартов обеспечивает возможность доступа к любым документам в любое время. Вместо транслирования текстовых документов «сверху-вниз» от менеджеров по продукту к архитекторам, затем к разработчикам, затем к инженерам по тестированию и т.д., все участники процесса разработки могут иметь доступ к необходимой информации в любой момент времени.

Такой виртуальной репозиторий обеспечивает один общий источник обновленной информации, доступ к которому не зависит от физического местонахождения пользователя. Более того, люди могут обмениваться информацией, опытом и идеями, принимать решения и проводить мозговые штурмы, почти как если бы все они вместе работали в одном офисе.

Почему "виртуальный" репозиторий?

Напрашивается вопрос - почему мы говорим о *виртуальном* репозитории, а не о *реальном*? Существует много причин, по которым это может быть лучше, чем «утрамбовывать» всю информацию в одну базу данных.

- Во-первых, создать "универсальную базу" всех данных, которые только могут вам понадобиться, сама по себе задача не из простых.
- Во-вторых, вы будете вынуждены поддерживать самые различные инструменты самых разных вендоров, которые существуют сейчас или могут появиться в будущем.
- А в третьих, одна массивная и громоздкая база данных, доступ к которой будет осуществляться из любой точки земного шара, сможет обеспечить хорошую производительность только для некоторых – для остальных это может быть кошмаром. И это не говоря еще о вопросах обновления версий, устранения неполадок (сети или оборудования) и прочих проблем, присущих одной физической базе данных.

Виртуальная база данных определенно лучше: распределенное расположение, оптимизация производительности, оптимизированные настройки под хранение данных каждого инструмента, простота обновления, простота сопряжения с другими базами.

Предупреждение: В попытке собрать все в одном месте вы становитесь заложником одного отдельного решения и одного поставщика, устраняя возможность выбора, свойственного потребителю.

Упрощение интеграции инструментов

Каким бы хорошим и удобным не казалось совместное использование ресурсов и активов с помощью виртуального репозитория, - в реальности это не так просто, как кажется. Инструменты собственного производства, проекты с привлечением сторонних компаний, множество вендоров – все это может создавать барьеры в виде несовместимых форматов данных и прочих технических факторов.

За последние несколько лет производители популярных инструментов, поддерживающих жизненный цикл разработки систем, типы которых приведены в таблице 6-1, наконец объединились, чтобы найти методы упрощения интеграции таких инструментов. Было создано сообщество Open Services for Lifecycle Collaboration (OSLC), целью которого является стимулирование новых форм взаимодействия и устранение барьеров между инструментами.

Положив в основу архитектуру интернета, OSLC устанавливает набор гибко связанных стандартов, общих форматов и служб, созданных для упрощения обмена и использования совместных ресурсов. OSLC облегчает использование комбинаций различных инструментов любых поставщиков, определяя общий доступ к данным жизненного цикла, например, к требованиям, запросам на изменение, тестам и их результатам, дефектам.

Таблица 6-1 Ключевые инструменты системной инженерии

Тип инструмента	*Основные возможности*
Управление требованиями и трассируемость	Сквозная динамическая трассировка, связывающая источник информации, целевое назначение системы и требования к системе / подсистеме
Разработка систем на основе моделей	Моделирование требований, функциональности системы, вариантов реализации, исследование затрат, исполнение моделей, валидация
Управление изменениями и конфигурациями	Управление взаимодействием, изменениями, общим репозиторием и конфигурацией
Автоматическое создание документов	Создание документов, содержащих информацию о требованиях, моделях, дизайне, спецификациях
Интегрированная системная и программная инженерия	Обеспечение интеграции между артефактами (сверху-вниз) – требованиями, моделями, разработкой встроенных приложений

Автоматическое создание документации

Даже в среде разработки на основе моделей необходимо ведение документации и создание отчетов для выполнения контрактных обязательств, подтверждения соблюдения норм, согласования, технического анализа и управления проектом. Зачастую документирование подразумевает большое количество рутинной ручной работы, например, извлечение конкретной диаграммы из инструмента моделирования или создание снимка экрана и их последующая вставка в документ. Более того, много времени тратится на сбор информации из различных источников, на подтверждение того, что это последняя версия данных, на их анализ и переформатирование - и все это лишь только для того, чтобы создать требуемый отчет.

Инструменты, автоматизирующие создание документов, значительно упрощают производство разнообразных форм отчетности, одновременно обеспечивая непротиворечивость и полноту представляемых сведений.

Такого рода инструменты автоматизации обеспечивают доступ к центральному репозиторию, идентифицируют требуемый тип информации, сортируют ее и создают отчет. Это упрощает не только сам процесс создания отчетов, но и возможность повторного использование информации, а также обеспечивает ее полноту и согласованность. Более того, у вас есть возможность консолидировать важную информацию из множества различных источников (даже из инструментов разных вендоров) в единый сводный отчет.

Разумеется, экономия времени при создании документов стоит на первом месте, но полное осознание пользы инструмента ощущается в тот момент, когда у вас что-то меняется. Стоит вам внести изменение в требования или модель и нажать кнопку, - и готово! - вы получаете уже обновленные документы.

Глава 7

10 способов стать лидером с помощью системной инженерии

В этой главе

- Устранение недостатков дизайна на ранних стадиях разработки
- Разработка гибких бизнес-моделей
- Контроль над сложным встроенным программным обеспечением
- Оптимизация управления требованиями
- Повторное использование кода для ускорения выпуска продукта
- Оптимизация разработки с помощью инструментов для взаимодействия
- Своевременная поставка на рынок сложных решений
- Использование интегрированной платформы разработки
- Тестировать раньше, тестировать чаще
- Совместная работа с требованиями для повышения эффективности разработок

Системная инженерия может дать большое конкурентное преимущество так необходимое вам, чобы преуспеть в разработке умных продуктов, которые имеют реальную материальную значимость для ваших заказчиков (и для вас тоже). Делая системную инженерию частью ваших ключевых бизнес-процессов, вы в большей степени обеспечиваете производство таких продуктов, которые на самом деле необходимы людям, - пусть даже с некоторыми (позже устранимыми) недостатками, зато отвечая динамике рынка.

В этой главе вы познакомитесь с десятью примерами того, как реальные компании адаптировали лучшие практики системной инженерии, и получили реальные ощутимые результаты их применения.

Устраняем недостатки дизайна, не дожидаясь эффекта лавины

Самой трудной миссией при разработке интеллектуальных продуктов является выявление недостатков на ранних стадиях процесса разработки. Не секрет, что большинство основных ошибок закрадывается в продукт на стадии формирования дизайна, и часто их невозможно обнаружить даже на фазе тестирования или, - что еще хуже — обнаруживаются они после того, как продукт пошел в производство.

Если для дорогостоящих товаров массового производства исправление ошибок дизайна может обойтись чрезвычайно дорого, то наличие ошибок в широко используемых системах обороны становится просто опасным. Стоит при этом заметить, что любые оборонные системы состоят из множества сложных подсистем, разработкой и производством которых занимается масса авторизованных субподрядчиков.

Субподрядчик Министерства обороны США компания Brockwell Technologies, расположенная в городе Хантсвилл штата Алабама, занимается созданием встраиваемых приложений реального времени для систем вооружения, а также встраиваемых систем диагностики военного транспорта. Разработка сопряженных и взаимодействующих систем вооружения ставит перед компанией сложную задачу: как внести изменения в одну из систем, не нарушив при этом работу других систем и всего комплекса.

Чтобы снизить вероятность внесения дефектов во взаимодействие подсистем, Brockwell Technologies внедрила в свои бизнес-процессы системное проектирование и тестирование на основе моделей. Моделируя одновременно структуру и поведение системы, инженеры Brockwell получили возможность создавать прототипы сложных систем и визуализировать насколько хорошо они будут функционировать. Такая прогнозная техника моделирования позволяет инженерам находить проблемные места и устранять их еще до того, как система уходит в производство.

Инвестиции Brockwell в системную инженерию одновременно играют на руку и Министерству обороны, и самой компании: Brockwell не только повысила надежность и безопасность создаваемых ею систем вооружения, но при этом на 40% снизила время вывода продукции на рынок.

Разработка гибких бизнес-моделей

Если вы планируете начать новый бизнес или войти на новый для себя рынок, то самая правильная вещь, которую вы можете сделать,

это разработать гибкую инфраструктуру бизнеса для воплощения принципов и подходов системной инженерии. Именно это сделала компания из Атланты несколько лет назад, выходя на раскаленный и перенасыщенный рынок передовых телекоммуникаций.

Опередить ветеранов индустрии оказалось возможным благодаря такой бизнес-структуре производства, которая позволила быстро адаптировать модель деловых отношений и предоставления услуг. Компании удалось перехватить инициативу (прежде, чем последовала их реакция) у консервативных специализированных поставщиков телекоммуникационных услуг и за счет гибкого построенного на открытой технологии производственного процесса обеспечить быстрое предоставление новых сервисов.

Компания пересмотрела все свои ключевые бизнес-процессы, охватывающие дирекцию, операционный и производственные отделы, ставя во главу угла одну цель: максимальная скорость предоставления новых услуг. Для создания новых услуг компания использовала единую платформу разработки систем, а также комплекс инструментов для совместной работы. Инфраструктура разработки систем компании обеспечивала управление и контроль за промежуточными рабочими продуктами и конечными поставками, поддерживала хранение результатов в центральном репозитории, облегчала взаимодействие и координацию распределенных подразделений, обеспечивала контроль версий и быстрое распространение обновлений.

Комбинация открытых систем, гибких процессов и эффективной совместной работы дали компании возможность поставлять на рынок новые сервисы менее, чем за 30 дней. И даже несмотря на то, что основной фокус в сфере телекоммуникаций продолжает смещаться от технологий как таковых в сторону оказания инновационных услуг, использующих новые способы передачи данных, компания сумела обеспечить себе лидирующие позиции в отрасли.

Контроль над сложным встроенным программным обеспечением

Умные и опытные компании знают, что если их продукт является составной частью большой «системы систем», то гарантированное качество продукта является наиболее важным фактором. В конце концов никому не хочется, чтобы его компания несла славу самого слабого звена в системе. Но по мере роста размеров и сложности встраиваемого программного обеспечения становится все трудней обеспечить должный уровень качества.

Долгое время «ручная» разработка программного обеспечения была основным источником доходов немецкого поставщика услуг,

который специализировался в сфере контрольно-измерительных технологий и инженерии процессов. Но когда компания приступила к разработке сложного встраиваемого программного обеспечения для систем удаленного контроля и управления фотоэлектрическими устройствами, возникла необходимость в использовании новой программной среды.

Исходя из четырех основных целей, – снижение числа дефектов, улучшение трассируемости, увеличение повторного использования программных модулей и обеспечение должного уровня качества продукта, — компания выбрала платформу разработки на основе моделей для системной инженерии реального времени и встроенных систем. Новая система обеспечивает традиционно высокий уровень качества за счет обнаружения и устранения проблем благодаря возможности тестировать модели еще на ранних стадиях дизайна. Дополнительно появилась возможность создания повторно используемых модулей исходного кода и подсистем, и как следствие – новые конкурентные преимущества.

Повышение эффективности с помощью управления требованиями

Чтобы создать правильный продукт — и создать его правильно, — необходимо четкое и однозначное понимание того, что нужно вашему заказчику и всем заинтересованным сторонам, а затем на базе этих потребностей тщательное формирование требований к системе. Без эффективного управления требованиями легко упустить из виду не только цели, но и ваших заказчиков.

Здесь уместно привести пример одной австралийской компании, которая, имея сотни географически распределенных разработчиков, многие из которых использовали различные инструменты для управления требованиями, боролась с неэффективными методами разработки и недостаточной трассируемостью требований. Руководство компании все больше стал беспокоить тот факт, что компания не могла тестировать требования в унифицированном виде, поскольку каждая из участвующих в разработке команд использовала свой собственный метод управления требованиями. Но хуже всего было то, что отсутствие желанной прозрачности увеличивало вероятность появления ошибки, что в конце концов могло поставить под угрозу сотрудничество с самым крупным заказчиком компании – вооруженными силами Австралии.

Для исправления ситуации и в целях улучшения анализа требований было решено в рамках всей компании внедрить единую систему управления требованиями. Обеспечивая доступ к централизованному репозиторию требований из любой точки, такое решение устраняло неразбериху и путаницу, затраты на исправления и переделку,

дублирующие друг друга разработки. Более того, это решение значительно облегчало полную трассируемость требований, что, в конечном итоге, придавало уверенность в том, что продукт, выезжающий за ворота компании, действительно отвечает нуждам заказчика.

Повторное использование кода для ускорения выпуска продукта

Если портфель вашей компании содержит линию продуктов со сходными базовыми возможностями, то, вероятнее всего, у вас накопилось множество сходного программного кода. Успешные компании структурируют имеющийся код в повторно используемые универсальные модули, которые упрощают разработку новых продуктов.

Именно эту стратегию взяла на вооружение компания Océ N.V., приступая к разработке самого быстрого принтера (печатающего на листовой бумаге). Будучи лидером на рынке технологий и услуг для управления цифровыми документами, Océ занимается разработкой программных приложений, обеспечивающих отправку в архив или на печать документов и данных через локальные сети и через Интернет.

Обычно для каждого нового принтера Océ код писался заново, но столкнувшись со сложнейшей задачей по согласованию кода для 17 процессоров, распределенных по разным компонентам нового принтера, руководство компании пришло к решению о пересмотре процесса разработки.

Компания Océ внедрила инструмент разработки на основе моделей, с помощью которого декомпозировала систему принтера на небольшие и более простые по дизайну подсистемы. Это дало возможность разработчикам компании смоделировать ряд параллельно действующих конечных автоматов, которые затем были превращены в программные модули для повторного использования во множественных и различных вариантах рабочего оборудования. При необходимости внесения изменений компания просто корректирует модели и обновляет код, кардинально снижая затраты времени на обработку запросов на изменение.

Сегодня более 50% программного кода компании Océ является повторно используемым, что существенно повышает эффективность и качество разработок. Фактически Océ смогла выпустить рабочий прототип нового принтера уже через 2 месяца — что было на 6 месяцев быстрее, чем раньше.

Оптимизация разработки с помощью инструментов для взаимодействия

Компании, которые создают сложные интеллектуальные и высокотехнологичные продукты, хорошо знают, что успех в той степени зависит от возможностей управления работами, в которой это позволяет организовать высокоуровневая инженерия. При отсутствии дисциплины системного уровня, координирующей усилия разрозненных инженерных групп, сложный продукт будет «иметь хорошую прессу» лишь с точки зрения его полного провала.

На стартовом этапе разработки автомобиля Chevy Volt компания General Motors (GM) затратила существенные усилия на внедрение лучших практик системной инженерии. GM внимательно проанализировала имеющиеся у нее практики разработки и методики управления техническим производством с целью их возможного улучшения.

Традиционно системные инженеры GM основное время тратили на сверку того, что поступало на вход разработчикам, чтобы убедиться, что каждый из них работает с одинаковыми версиями требований или других документов. Чтобы освободить инженеров от этой рутинной работы и сфокусировать их на более важных аспектах разработки – соблюдении функциональности и качества, – GM решила внедрить коммерческий инструмент для организации координации разных команд разработчиков, а также для управления единой версией требований и другой документации.

Помимо этого GM внедрила практики разработки систем на основе моделей, чтобы управлять сложностью (функционирование Volt поддерживается 10 миллионами строк кода). Для визуализации процессов взаимодействий между встроенными системами разработчики GM использовали модели, которые затем исполнялись, чтобы тестировать их поведение.

Сочетание инструментов совместной разработки и методов разработки систем на основе моделей реально окупило все усилия. Работа над Volt была завершена всего за 29 месяцев — рекорд для компании, которой для выпуска новой модели обычно требовалось не менее 5 лет.

Своевременная поставка на рынок сложных решений

Чтобы преуспеть в сегодняшних суровых условиях конкуренции, те компании, которые разрабатывают большие и сложные системы,

должны максимально сфокусировать свой процесс разработки на управлении требованиями. Возможность формировать и управлять требованиями может сыграть ключевую роль в получении крупного заказа.

Одна из оборонных компаний искала методы снижения риска по своевременной поставке на рынок сложнейшей по исполнению и многомиллионной по стоимости системы систем. В ходе поисков была предложена инновационная идея: свести вместе управление требованиями и инфраструктуру предприятия таким образом, чтобы обеспечить создание сложнейшей системы, разрабатываемой на основе требований, в установленные сроки.

Решение компании базировалось на доступных инструментах для управления требованиями и для планирования архитектуры систем. В итоге компания может теперь не просто определять технологические решения, но и формировать эксплуатационные, технические, обучающие и системные решения — в рамках всего жизненного цикла разработки. Результатом стало значительное ускорение выпуска больших и сложных, высококачественных и высокотехнологичных систем. И, что еще более важно, если заказчик вносит изменения в требования, то компания может очень быстро продемонстрировать на что могут повлиять эти изменения.

Повышение производительности при использовании интегрированной платформы разработки

От компаний, которые разрабатывают системы с высокими требованиями к безопасности, часто требуют подтвердить, что их процесс разаботки соответствует национальным и международным стандартам безопасности. Если имеющаяся у компании среда разработки фрагментирована, то ей бывает очень трудно доказать такое соответствие.

Подобная ситуация сложилась в одной австралийской компании, у которой команды разработки функционировали в рамках двух крупных площадок и при этом использовали различные наборы инструментов, и которая решила провести ряд изменений для успеха новых проектов. Компания занимается созданием средств сигнализации и управления на железных дорогах с высокими требованиями к безопасности. При этом заказчикам компании требовалось обеспечить совместимость новых разработок с их уже существующими системами.

Компания нуждалась в интегрированной среде разработки, которая бы обеспечила соответствие международным стандартам

безопасности и надежности. Компания внедрила единое решение для управления требованиями и конфигурацией, которое позволило не только удовлетворять все требования заказчиков, но и создать систему четкого взаимодействия распределенных команд разработчиков.

Тестировать раньше, тестировать чаще

Если ваши заказчики будут как можно раньше и чаще «держать в руках» либо прототип, либо исполняемую модель, это будет иметь огромное значение. Чем раньше вы получите отзывы о соответствии ваших идей мыслям заказчика, тем меньшее количество проблем ожидает вас в дальнейшем.

Раннее тестирование – пусть даже в упрощенном виде - окупается на этапе завершающих испытаний. Предоставляя командам тестировщиков ранний доступ, - даже если они знают, что продукт еще реально не готов («сырой») или что это всего лишь прототип, - вы не только помогаете тестировщикам улучшить планы и процедуры тестирования, но и оказываете неоценимую услугу дизайнерам, давая им возможность на ранних стадиях разработки убедиться в правильности своих идей.

Совместная работа с требованиями позволяет сохранить и время, и деньги

Очень часто в крупных корпорациях две или даже более географически рапределенные команды разработки трудятся над параллельными релизами с общими требованиями к ним. Большое количество времени и усилий можно было бы сэкономить, если у этих команд был бы механизм, поддерживающий совместный доступ к требованиям.

Обратимся за примером к компании из штата Мичиган, являющейся ведущим поставщиком мобильных электронных устройств и транспортных систем. Компании требовалось улучшить методы коммуникации и взаимодействия внутри ее глобальных команд разработки при работе над параллельными релизами с общими требованиями к ним.

Внедрение инструмента для управления требованиями значительно облегчило разработчикам задачу совместного использования требований. Разработчики могли теперь импортировать требования из общего репозитория, чтобы повторно использовать их в новом проекте, избегая ненужного и затратного дублирования усилий, что экономило время и снижало затраты на разработку. Применение инструмента для управления требованиями, обеспечивающего последовательное и усовершенствованное взаимодействие, дало компании возможность поставлять на рынок продукты высокого качества и в значительно меньшие сроки.

Дополнительные сведения см. в книгах:

"Rational Harmony for Systems Engineering Deskbook Rel. 3.1.2 Model-based Systems Engineering with Rational Rhapsody," Д-р Ханс-Питер Хофман (Hans-Peter Hoffmann). Сайт IBM Corporation, Software Group.

"Model Driven Systems Development with Rational Products Redbook". Авторы: Brian Nolan, Barclay Brown, Laurent Balmelli, Tim Bohn и Ueli Wahli. Издательство: International Technical Support Organization.

Понимание проблем, которые призвана решать системная инженерия

Системная инженерия - это междисциплинарный подход к созданию крупных сложных систем, которые соответствуют определенному набору экономических и технических требований. В аэрокосмической и оборонной промышленности системная инженерия используется уже давно и многие из полученных опытным путем знаний уже применяются и в других сферах. Автомобили, телефоны, телевизоры становятся все "умнее", и для их производства требуются технологии эпохи покорения космоса. Книга *"Системная инженерия для «чайников»"* (ограниченная серия от IBM) поможет вам добраться до сути тематики, разобраться в методах решения проблем и создать успешное производство.

- ***Интеллектуальный продукт* — это принципиально новый подход к разработке систем**
- ***Понимание процесса разработки систем* позволяет получить общее верхнеуровневое представление о системной инженерии**
- ***Моделирование* позволяет лучше понять структуру системы и ее поведение**
- ***Улучшение сотрудничества* позволит объединить команды разработчиков**

В этой книге вы найдете:

- **примеры, связанные с реальными компаниями, чей успех обусловлен использованием системной инженерии**
- **как создать правильную систему, используя валидацию и верификацию**
- **методы создания интеллектуальных продуктов**

Делая все проще!

WILEY

ISBN: 978-1-118-83277-6
IBM, серия: RAM14002RURU