Федеральное агентство по образованию
Сибирский федеральный университет

АНАЛИЗ ТРЕБОВАНИЙ К ИНФОРМАЦИОННЫМ СИСТЕМАМ

Конспект лекций

Маглинец Ю.А.

Красноярск
СФУ
2007

1. Введение. Понятие и классификация требований

План лекции

Определение ИС
Классификация ИС
Классификация по масштабу
Классификация по архитектуре
Классификация по характеру использования информации
Классификация по системе представления данных
Классификация по поддерживаемым стандартам управления и технологиям коммуникации
Классификация по степени автоматизации
Роль требований в задаче внедрения АИС
Определение понятия требования
Классификация требований
Требования к продукту и процессу
Уровни требований
Системные требования и требования к программному обеспечению
Функциональные, нефункциональные требования и характеристики продукта
Классификация RUP
Методологии и стандарты, регламентирующие работу с требованиями

Определение ИС

Далее по тексту *информационной системой (ИС)*[1], либо автоматизированной ИС, *АИС,* будем называть программно-аппаратную систему, предназначенную для автоматизации целенаправленной деятельности конечных пользователей, обеспечивающую, в соответствие с заложенной в неё логикой обработки, возможность получения, модификации и хранения информации.

Ключевым моментом в этом определении является понятие «целенаправленной деятельности». Речь идёт о деятельности, направленной на решение конкретной задачи, стоящей перед пользователем (коллективом пользователей).

Некоторые исследователи (см., например, [1]) определяют ИС несколько иным образом. *ИС в широком смысле* – взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели.

Основные отличия такого подхода: 1) ввод пользователей системы «внутрь» ИС, 2) необязательность использования средств вычислительной техники. Такой подход также имеет право на жизнь. Так, например, в нём удобно прослеживать общую историю возникновения и развития систематических средств обработки информации в бизнесе, которая началась, очевидно, в докомпьютерную эпоху. Однако, так как целью нашего курса является изучение анализа требований к компьютерным системам обработки информации, будем пользоваться приведённым выше первым определением.

Рассмотрим **примеры** некоторых программных средств, являющихся, либо не являющихся ИС.

- 1С-Бухгалтерия 8.0. Используется в целях формирования бухгалтерской отчётности предприятия перед налоговыми органами. Является информационной системой.

---

[1] Маглинец Ю.А. Разработка информационных систем. Часть 1, Структурные методы. – Красноярск.: Кларитеанум, 2004. – 120 с.

- MS Excel. Программное средство универсального характера, предназначенное для манипуляций с данными, представленными в табличной форме автоматизации расчётов, формирования разнообразных диаграмм для анализа данных. Не является информационной системой.
- Книга MS Excel, содержащая сведения о штатном расписании, работниках предприятия и оснащённая макросами, позволяющими рассчитывать заработную плату и формировать платёжные ведомости. Является информационной системой.
- Система Axapta Retail комплексной автоматизации деятельности сети розничных магазинов. Является информационной системой.
- Реляционная база данных DB-2 фирмы IBM. Не является информационной системой.

Классификация ИС

Информационные системы могут быть классифицированы по различным признакам. Рассмотрим наиболее важные из них.

Классификация по масштабу

По масштабу ИС будем подразделять на однопользовательские, групповые и корпоративные.

Однопользовательские ИС, как это ясно из названия, предназначены для использования на одном рабочем месте. В настоящее время на мировом и отечественном рынке представлено множество решений, предназначенных для автоматизации деятельности отдельно взятого пользователя. Как правило, это – решения, ориентированные на специалиста в той или иной области, будь то составление спецификаций для сборки изделий из комплектующих, планирование ремонтов оборудования, учёт расходов и доходов частного предпринимателя оптовой торговли, либо составление расписаний занятий в деканате.

В настоящее время альтернативу таким узкоспециализированным системам составили табличные процессоры, не имеющие проблемной специализации, в первую очередь – MS Excel. Системы этого класса трудно отнести к классу ИС, но зачастую они позволяют непрограммирующему специалисту создать и, что очень важно, самостоятельно развивать собственные решения, заменяющие, а местами и перекрывающие функционал однопользовательских систем образца 90-х годов.

В основе большинства однопользовательских систем лежит стандарт X-Base (Clipper, FoxPro, dBase). Широко используются также решения на базе систем Paradox, Clarion, MS Access. Каждая из перечисленных конкурирующих систем обладает собственной высокоуровневой инструментальной средой, позволяющей спроектировать базу данных, логику обработки, пользовательский интерфейс, отчёты с помощью «помощников»-построителей. На рубеже тысячелетий появились также и однопользовательские решения на базе промышленных реляционных СУБД. В этом случае ПО сервера инсталлируется непосредственно на рабочую станцию пользователя. Примером может служить Personal Oracle. Данные решения предъявляют значительные требования к ресурсам рабочей станции, однако несут в себе многие преимущества промышленных СУБД.

Групповые системы предназначены для автоматизации деятельности в рабочей группе (отделе, кластере, группе проекта и т.д.). В отличие от однопользовательских ИС, групповые системы, как правило, представляют специализированные клиентские решения (их часто называют автоматизированными рабочими местами, АРМ) для различных участников группы. Например, для оптовой фирмы, ИС может представлять набор таких АРМ, как «Менеджер по продажам», «Кладовщик», «Снабженец», «Директор». Для учебного планирования – «Преподаватель», «Работник бюро

планирования», «Работник учебного отдела», «Специалист по планированию на кафедре», «Работник деканата».

Групповое использование решений на базе табличных процессоров возможно, но имеет существенные ограничения, связанные с разграничением доступа, регламентацией и синхронизацией вносимых изменений. По сути единственный режим их использования, обеспечивающий корректность данных – «файловый сервер, один автор, N читателей».

При создании групповых ИС в целом используются те же средства и инструментальные среды, что и при создании однопользовательских ИС. Следует, однако, отметить, что для использования в группе при выборе между системами с файловым и реляционным сервером следует отдавать предпочтение реляционному серверу, причём целесообразно использование выделенного сервера. Это может быть, например, сервер Oracle, DB2, MS SQL, Sybase, Informix.

Корпоративные ИС (КИС) предназначены для автоматизации деятельности предприятия. В англоязычной литературе понятие «КИС» неразрывно связано с понятием «ERP» (Enterprise Resource Planning). В основе ERP-систем лежит международный стандарт управления предприятием MRP-II (Manufacture Resource Planning), обеспечивающий возможность учета, анализа и планирования основных ресурсов - финансов, человеческих, материальных. Соответственно, корпоративные ERP-системы – набор интегрированных приложений, которые комплексно, в едином информационном пространстве поддерживают все основные аспекты управленческой деятельности предприятий: планирование ресурсов (финансовых, человеческих, материальных) для производства товаров (услуг), оперативное управление выполнением планов (включая снабжение, сбыт, ведение договоров), все виды учета и анализ результатов хозяйственной деятельности.

Среди требований, предъявляемым к современным КИС:

- централизация данных в единой базе (в основе – всегда промышленная СУБД),
- близкий к реальному времени режим работы,
- сохранение общей модели управления для предприятий разных отраслей,
- поддержка территориально-распределенных структур,
- работа на широком круге аппаратно-программных платформ и СУБД.

Примеры ERP-систем – SAP R3, «Галактика», MS Navision Axapta.

Классификация по архитектуре

1. Архитектура «Файл-сервер». Исторически первая архитектура информационных систем. Как исполняемые модули, так и данные размещаются в отдельных файлах операционной системы. Доступ к данным осуществляется путём указания пути (path) и использования файловых операций (открыть, считать, записать). Для хранения данных используется выделенный сервер (отдельный компьютер), который и является файловым сервером. Исполняемые модули хранятся либо на рабочих станциях, либо на файловом сервере. В последнем случае упрощается процедура их администрирования, но при этом возрастают требования к надёжности сети.

2. Архитектура «Клиент-сервер». Клиент-сервер – это не только архитектура, это – новая парадигма, пришедшая на смену устаревшим концепциям. Суть её заключается в том, что клиент (исполняемый модуль) запрашивает те или иные сервисы в соответствии с определённым протоколом обмена данными. При этом, в отличии от ситуации с файловым сервером, нет необходимости в использовании прямых путей операционной системы: клиент их «не знает», ему «известны» лишь имя источника данных и другие специальные сведения, используемые для авторизации клиента на сервере. Сервер, который физически может находиться на том же компьютере, а может - на другом конце земного шара, обрабатывает запрос клиента и, произведя соответствующие манипуляции с данными, передаёт клиенту запрашиваемую порцию данных.

В рамках направления «клиент-сервер» существуют два основных «диалекта»: «тонкий» и «толстый» клиент.

В системах на основе тонкого клиента используется мощный сервер баз данных, это – высокопроизводительный компьютер и библиотека так называемых хранимых процедур, позволяющих производить вычисления, реализующие основную логику обработки данных, непосредственно на сервере. Клиентское приложение, соответственно, предъявляет невысокие требования к аппаратному обеспечению рабочей станции. Основное достоинство таких систем – относительная дешевизна клиентских станций.

Системы с толстым клиентом, напротив, реализуют основную логику обработки на клиенте, а сервер представляет собой в чистом виде сервер баз данных, обеспечивающий исполнение только стандартизованных запросов на манипуляцию с данными (как правило – чтение, запись, модификацию данных в таблицах реляционной базы данных). В системах такого класса требования к рабочей станции выше, а к серверу – ниже. Достоинство архитектуры – переносимость серверной компоненты на серверы различных производителей: все промышленные серверы баз данных реляционного типа поддерживают работу со стандартизованным языком манипулирования данными SQL, но внутренний встроенный язык обработки данных, необходимый для реализации логики обработки на сервере у каждого из серверов свой.

3. Трёхслойная архитектура. Базируется на дальнейшей специализации компонент архитектуры: клиент занимается только организацией интерфейса с пользователем, сервер баз данных – только стандартизованной обработкой данных. Для реализации логики обработки данных архитектура предусматривает отдельный слой – слой бизнес-логики. Этот слой может представлять собой либо выделенный сервер (сервер приложений), либо размещаться на клиенте в качестве динамической библиотеки. Данная архитектура позволила соединить достоинства тонкого и толстого клиентов: хорошая переносимость соединяется в ней с невысокими требованиями к клиенту.

С развитием интранет-интернет технологий появилась разновидность трёхслойной архитектуры на основании использования web-технологий. В этой разновидности роль сервера приложений играет web-сервер, а в качестве клиента используется стандартный web-браузер. Достоинства – в пониженных требованиях к клиенту и в легкой встраиваемости данной архитектуры в мировые информационные сети. Основной недостаток - известные ограничения, накладываемые на интерфейс пользователя web-браузерами.

Классификация по характеру использования информации

С некоторой степенью приближения все ИС можно разделить на 2 класса: информационно-поисковые и управляющие.

Конечные пользователи информационно-поисковых систем (ИПС), как правило, имеют доступ к хранимым данным только «по чтению» и используют данные системы для поиска ответов на те или иные вопросы. Доступ по модификации данных имеет администратор системы, в функции которого входит обеспечение актуальности информации, устранение ошибок.

Классические примеры ИПС – системы поиска в библиотеках, на транспорте (справки о наличии билетов). На современном этапе развития информационных технологий классические ИПС постепенно вытесняются поисковыми серверами Интернет – общего назначения и специализированными.

Альтернатива ИПС – управляющие системы автоматизируют (полностью или частично) деятельность, связанную с принятием решений. Действия конечных пользователей таких систем приводят к модификации информации, что, конечно, не исключает возможности просто получать информацию, как в ИПС.

Примеры управляющих систем – системы бухгалтерского учета, системы планирования производственных ресурсов и т.п.

Классификация по системе представления данных

Среди наиболее распространённых средств и моделей представления данных следует выделить:

- «самодельные» форматы хранения данных, хранящихся в файлах (текстовых, бинарных);
- специализированные форматы хранения данных, использовавшиеся в «дореляционный» период (например, x-Base, paradox);
- языки структурированной разметки на основе формата xml;
- реляционная модель; SQL сервер;
- объектная, объектно-реляционная модель;
- документоориентированное хранилище (IBM Lotus/Domino).

Классификация по поддерживаемым стандартам управления и технологиям коммуникации

Эпоха стихийной разработки АИС закончена. Современные автоматизированные информационные системы разрабатываются, исходя из сложившихся реалий автоматизированного управления бизнесом. Существует значительное количество концепций, технологий, подходов, нашедших своё эффективное применение в различных отраслях промышленности по всему миру. Некоторые из них приобрели статус международных стандартов. В спецификации АИС, разрабатываемой для массовой продажи, как правило, указывается – какие стандарты и технологии управления она поддерживает. Менее строги требования к АИС, создаваемым под заказ для конкретного предприятия. Однако и в этом случае не учитывать сложившийся в мире позитивный опыт просто неразумно. Ниже перечислены некоторые, наиболее важные, технологии и стандарты.

MRP (Material Requirements Planning) – планирование поставок материалов, исходя из данных о комплектации производимой продукции и плана продаж.

CRP (Capacity Requirements Planning) – планирование производственных мощностей, исходя из данных о технологии производимой продукции и прогноза спроса.

MRPII (Manufacture Resource Planning) – планирование материальных, мощностных и финансовых ресурсов, необходимых для производства. Стандартизовано ISO.

ERP (Enterprise Resource Planning) – финансово-ориентированное планирование ресурсов предприятия, необходимых для получения, изготовления, отгрузки и учёта заказов потребителей на основе интеграции всех отделов и подразделений компании.

SCM (Supple Chain Management) – управление цепочками поставок. Реализация бизнес-процессов на базе внешних предприятий и торговых площадок Основано на референтной модели SCOR, стандартизованной Supply Chain Council.

CRM (Customer Relationship Management) - управление взаимоотношениями с заказчиками. Комплекс методов и средств, нацеленный на завоевание, удовлетворение требований и сохранение платежеспособных клиентов.

ERPII (Enterprise Resource & Relationship Processing) - управление ресурсами и взаимоотношениями предприятия. Объединяет в себе 3 вышеперечисленные технологии.

Workflow – технология, управляющая потоком работ при помощи программного обеспечения, способного интерпретировать описание процесса, взаимодействовать с его участниками и при необходимости вызывать соответствующие программные приложения.

OLAP (Online Analytical Processing) – оперативный анализ данных. Технология поддержки принятия управленческих решений на основе концепции многомерных кубов информации.

Project Management – управление проектами. Поддерживается рядом международных стандартов.

CALS (Continuous Acquisition and Lifecycle Support) — непрерывная информационная поддержка поставок и жизненного цикла. Описывает совокупность

принципов и технологий информационной поддержки жизненного цикла продукции на всех его стадиях. Объединяет в себе практически все вышеперечисленные подходы и технологии.

Столь лаконичные определения, конечно же, позволяют лишь ознакомиться с современной терминологией. Задача их детального изучения не входит в план занятий. Тех, кто захочет получить подробный материал по этим вопросам, можно порекомендовать следующие литературные источники [2-6].

Классификация по степени автоматизации

Приводится классификация из [1].

Ручные ИС характеризуются отсутствием современных технических средств переработки информации и выполнением всех операций человеком. Например, о деятельности менеджера в фирме, где отсутствуют компьютеры, можно говорить, что он работает с ручной ИС.

Автоматические ИС выполняют все операции по переработке информации без участия человека.

Автоматизированные ИС предполагают участие в процессе обработки информации и человека, и технических средств, причем главная роль отводится компьютеру. В современном толковании в термин "информационная система", как правило, вкладывается понятие автоматизируемой системы.

Роль требований в задаче внедрения АИС

Какие можно сделать выводы из рассмотренной классификации АИС? Даже не говоря о многообразии производителей АИС, не рассмотренного в настоящей лекции, на основании только сформулированных признаков становится очевидным, что существует значительное количество АИС и данные АИС существенно различаются между собой. Следовательно, выбор АИС для предприятия – достаточно нетривиальная задача. Для того, чтобы успешно её решить, необходимо хорошо знать объект внедрения (автоматизируемое предприятие), особенности его деятельности, стратегию развития и многие другие аспекты, предопределяющие характеристики закупаемой АИС. Указанные знания в конечном итоге формализуются в документе требований к АИС, на основе которого и осуществляется выбор и последующая настройка АИС. В ещё большей степени требования к АИС важны при разработке АИС на заказ. Подробнее об этом – в следующих лекциях.

Определение понятия требования

Л.Новиков в русской редакции нотации RUP [15] приводит следующее определение: «Требование – это условие или возможность, которой должна соответствовать система».

В IEEE Standard Glossary of Software Engineering Terminology (1990) [7] данное понятие трактуется шире. Требование – это:

1. условия или возможности, необходимые пользователю для решения проблем или достижения целей;

2. условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам;

3. документированное представление условий или возможностей для пунктов 1 и 2 (конец цитаты).

Введём ещё одно определение. Требования – это исходные данные, на основании которых проектируются и создаются автоматизированные информационные системы. Первичные данные поступают из различных источников, характеризуются противоречивостью, неполнотой, нечёткостью, изменчивостью. Требования нужны в

частности для того, чтобы Разработчик мог определить и согласовать с Заказчиком временные и финансовые перспективы проекта автоматизации. Поэтому значительная часть требований должна быть собрана и обработана на ранних этапах создания АИС. Однако собрать на ранних стадиях все данные, необходимые для реализации АИС, удаётся только в исключительных случаях. На практике процесс сбора, анализа и обработки растянут во времени на протяжении всего жизненного цикла АИС, зачастую нетривиален и содержит множество подводных камней; подробнее о процессе – в лекциях 4-8.

Классификация требований

Существует значительное количество различных методов классификации требований [8-13], наиболее существенные из которых будут рассмотрены в лекции.

Требования к продукту и процессу

Требования к продукту. В своей основе требования – это то, что формулирует заказчик. Цель, которую он преследует – получить хороший конечный продукт: функциональный и удобный в использовании. Поэтому требования к продукту являются основополагающим классом требований. Более подробно требования к продукту детализируются в следующих ниже классификациях.

Требования к проекту. Вопросы формулирования требований к проекту, т.е. к тому, как Разработчик будет выполнять работы по созданию целевой системы, казалось бы, не лежат в компетенции Заказчика. Без регламентации процесса Заказчиком легко можно было бы обойтись, если бы все проекты всегда выполнялись точно и в срок. Однако, к сожалению, мировая статистика результатов программных проектов говорит об обратном. Заказчик, вступая в договорные отношения с Разработчиком, несёт различные риски, главными из которых является риск получить продукт с опозданием, либо ненадлежащего качества. Основные мероприятия по контролю и снижению риска – регламентация процесса создания программного обеспечения и его аудит.

Насколько подробно Заказчику следует регламентировать требования к проекту – вопрос риторический. Ответ на него зависит о множества факторов, таких, как ценность конечного продукта для Заказчика, степень доверия Заказчика к Разработчику, сумма подписанного контракта, увязка срока сдачи продукта в эксплуатацию с бизнес-планами Заказчика и т.д. Однако, со всей определённостью можно сказать следующее: 1) регламентация процесса Заказчиком позволяет снизить его риски; 2) мероприятия Заказчика по регламентации процесса приводят к дополнительным накладным расходам. Требуется найти разумный компромисс между степенью контроля рисков и величиной расходов.

В качестве требований к проекту могут быть внесён регламент отчётов Разработчика, совместных семинаров по оценке промежуточных результатов, определены характеристики компетенций участников рабочей группы, исполняющих проект, их количество, указана методология управления проектом. Ниже сформулирован пример формулировки требования к оффшорному проекту (Заказчик и Разработчик физически находятся в разных государствах) – в этой ситуации Заказчику требуется жёсткий контроль над Разработчиком.

1) Разработчик представляет Заказчику согласованный план работ с детализацией (WorkBreakdownStructure - WBS) с точностью до конкретных исполнителей.

2) Разработчик осуществляет ежедневные сборки, регрессионное тестирование компонент разрабатываемого продукта и тестирование продукта в целом.

3) Все управленческие и проектные артефакты, исходные коды и тестовые примеры размещаются в режиме online в интегрированной среде разработки Rational ClearCase® с возможностью для Заказчика осуществления online-мониторинга на базе web-технологий.

Уровни требований

Внедрение ИС на предприятии всегда преследует конкретные бизнес-цели – такие, как, например, повышение прозрачности бизнеса, сокращение сроков обработки информации, экономия накладных расходов и т.д. Современные информационные системы – это крупные программные системы, содержащие в себе множество модулей, функциональных, интерфейсных элементов, отчётов и т.д. Как охватить единым взором такие разнородные вещи, как цели, преследуемые топ-менеджером предприятия Заказчика, описание интерфейса пользователя и характеристики модуля, осуществляющего расчёт себестоимости изделия?

К счастью, человечество уже давно изобрело приёмы борьбы со сложностью, широко применяемые в моделировании сложных объектов – абстракцию и декомпозицию. Применительно к дисциплине анализа требований к программным системам эти принципы работают следующим образом. Требования разделяются по уровням. Уровни требований связаны, с одной стороны, с уровнем абстракции системы, с другой – с уровнем управления на предприятии.

Обычно выделяют три уровня требований.

На верхнем уровне представлены так называемые бизнес-требования (business requirements). Примеры бизнес-требования: система должна сократить срок оборачиваемости обрабатываемых на предприятии заказов в три раза. Бизнес-требования обычно формулируются топ-менеджерами, либо акционерами предприятия.

Следующий уровень – уровень требований пользователей (user requirements). Пример требования пользователя: система должна представлять диалоговые средства для ввода исчерпывающей информации о заказе, последующей фиксации информации в базе данных и маршрутизации информации о заказе к сотруднику, отвечающему за его планирование и исполнение. Требования пользователей часто бывают плохо структурированными, дублирующимися, противоречивыми. Поэтому для создания системы важен третий уровень, в котором осуществляется формализация требований.

Третий уровень – функциональный (functional requirements). Пример функциональных требований (или просто функций) по работе с электронным заказом: заказ может быть *создан*, *отредактирован*, *удалён* и *перемещён* с участка на участок.

Существуют объективные противоречия между требованиями различных уровней. Так, очевидным бизнес-требованием является требование о полноте информации, собираемой на рабочих местах пользователей в единую базу данных. Чем полнее информация – тем глубже база для анализа деятельности и принятия решений. С другой стороны, конкретному пользователю системы вполне может быть достаточно использования только той части информации, которая влияет на выполнение его основных функций.

Важные правила внедрения и использования АИС на предприятии – «Одна точка сбора», «Данные собираются там, где они появляются». Использование этих правил позволяет избежать затрат на необоснованное дублирование информации и, что важнее – потерь от ошибок учёта, неизбежно возникающих при дублировании точек ввода.

Внедрение АИС на предприятии приводит к необходимости оснащения всех точек ввода информации автоматизированными рабочими местами (АИС), обучению персонала и, зачастую, оптимизации и повышению уровня формализации рабочих процессов, выполняемых персоналом. Поэтому внедрение АИС – непростой процесс, часто требующий «перекройки человеческого материала» и встречающий сопротивление со стороны пользователей, которые не готовы, либо не хотят работать по-новому.

Системные требования и требования к программному обеспечению

Существуют различные трактовки понятия «Системные требования» (system requirements).

К. Вигерс формулирует данный термин, как «высокоуровневые требования к продукту, которые содержат многие подсистемы, то есть системе» [8]. При этом под системой понимается программная, программно-аппаратная, либо человеко-машинная система. Данная система является сложной, структурированной системой и системные требования являются подмножеством функциональных требований к продукту. В данное подмножество целесообразно относить наиболее важные, существенные требования, которые относятся в целом к системе и не содержат избыточной детализации.

INCOSE (International Council on Systems Engineering) даёт более детальное определение системы: «комбинация взаимодействующих элементов, созданная для достижения определенных целей; может включать аппаратные средства, программное обеспечение, встроенное ПО, другие средства, людей, информацию, техники (подходы), службы и другие поддерживающие элементы». Таким образом, происходит разделение между системными требованиями, как обобщающему понятию и требованиями к программному обеспечению, как выделенному подмножеству системных требований, направленных исключительно на программные компоненты системы. Этот же подход прослеживается в стандарте ГОСТ Р ИСО/МЭК 12207/99 [14]: работы, связанные с системой в целом и с программным обеспечением выделяются в отдельные группы в целях удобства оперирования.

В практике компьютерной инженерии бытует другой, более узкий контекст использования данного понятия: под системными требованиями в узком смысле понимается требования, выдвигаемые прикладной программной системой (в частности – информационной) к среде своего функционирования (системной, аппаратной). Пример таких требований – тактовая частота процессора, объём памяти, требования к выбору операционной системы.

Функциональные, нефункциональные требования и характеристики продукта

Функциональные требования регламентируют функционирование или поведение системы (behavioral requirements). Функциональные требования отвечают на вопрос «что должна делать система» в тех или иных ситуациях. Функциональные требования определяют основной «фронт работ» Разработчика, и устанавливают цели, задачи и сервисы, предоставляемые системой Заказчику.

Функциональные требования записываются, как правило, при посредстве предписывающих правил: «система должна позволять кладовщику формировать приходные и расходные накладные». Другим способом являются так называемые варианты использования (uses cases) – популярный и весьма продуктивный способ представления требований.

Это – основной, определяющий вид требований, который будет рассматриваться на протяжении всего лекционного курса.

Нефункциональные требования, соответственно, регламентируют внутренние и внешние условия или атрибуты функционирования системы. К. Вигерс [8] выделяет следующие основные группы нефункциональных требований:

- Внешние интерфейсы (External Interfaces),
- Атрибуты качества (Quality Attributes),
- Ограничения (Constraints).

Среди *внешних интерфейсов* в большинстве современных АИС наиболее важным является интерфейс пользователя (User Interface, UI). Кроме того, выделяются интерфейсы с внешними устройствами (аппаратные интерфейсы), программные интерфейсы и интерфейсы передачи информации (коммуникационные интерфейсы).

Основные *атрибуты качества*:

- Применимость,

- Надежность,
- Производительность,
- Эксплуатационная пригодность,

достаточно хорошо раскрыты в модели FURPS (см. ниже).

Ограничения [8] - формулировки условий, модифицирующих требования или наборы требований, сужая выбор возможных решений по их реализации. выбор платформы реализации и/или развертывания (протоколы, серверы приложений, баз данных, ...), которые, в свою очередь, могут относиться, например, к внешним интерфейсам (конец цитаты).

Характеристики продукта. К.Вигерс [8] формулирует характеристику, «фичу» (feature), как набор логически связанных функциональных требований, которые обеспечивают возможности пользователя и удовлетворяют бизнес-цели.

Существует и более общий взгляд на данное понятие[2]: «features могут быть как относящимся к функциональным, так и к нефункциональным требованиям и могут изменяться от версии к версии продукта».

С.Орлик в [12] отмечает, что «с точки зрения инженерии требований, features являются самостоятельным артефактом, который может быть соотнесен как с функциональными требованиями, так и с нефункциональными».

Роль характеристик проявляется в первую очередь в области маркетинга: не всякий потенциальный потребитель продукта станет читать его функциональные описания, а набор ключевых характеристик, характеризующих конкурентные преимущества, можно сделать лаконичным и уместить на одной странице рекламной листовки, либо напечатать на компакт-диске.

Классификация RUP

В спецификациях Rational Unified Process при классификации требований используется модель FURPS+ со ссылкой на стандарт IEEE Std 610.12.1990 [7].

Акроним FURPS обозначает следующие категории требований:

- Functionality (Функциональность)
- Usability (Применимость)
- Reliability (Надёжность)
- Performance (Производительность)
- Supportability (эксплуатационная пригодность).

Символ «+» расширяет FURPS-модель, добавляя к ней:

- ограничения проекта,
- требования выполнения,
- требования к интерфейсу,
- физические требования,

часть из которых уже была рассмотрена выше.

Кроме того, в спецификациях RUP выделяются такие категории требований, как

- требования, указывающие на необходимость согласованности с некоторыми юридическими и нормативными актами;
- требования к лицензированию,
- требования к документированию.

Методологии и стандарты, регламентирующие работу с требованиями

Среди основополагающих нормативных документов в области работы с требованиями можно выделить следующие.

1. Разработки IEEE:
   - IEEE 1362 “Concept of Operations Document”.

[2] Kurt Bittner, Ian Spence. Use Case Modeling, 2002.

- IEEE 1233 «Guide for Developing System Requirements Specifications».
- IEEE Standard 830-1998, «IEEE Recommended Practice for Software Requirements Specifications»
- IEEE Standard Glossary of Software Engineering Terminology/IEEE Std 610.12-1990
- IEEE Guide to the Software Engineering Body of Knowledge (1) - SWEBOK®, 2004.

2. Отечественные ГОСТ:
   - ГОСТ 34.601-90. Информационная технология. Автоматизированные системы. Стадии создания.
   - ГОСТ 34.602-89. Информационная технология. Техническое задание на создание автоматизированной системы
   - ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению.

Литература к лекции

1. Макарова Н.В. Информатика: Учебник. - М.: Финансы и статистика, 2003. - 768 с.
2. ERP системы. Современное планирование и управление ресурсами предприятия. Выбор, внедрение, эксплуатация/Дэниел О'Лири;[Пер. с англ. Ю.И.Водопьяновой]. – М.: ООО «Вершина», 2004. – 272 с.
3. Меняев М.Ф. Информационные технологии управления: Учебное пособие: В 3 кн.: Книга 3: Системы управления организацией. – М.: Омега-Л, 2003. – 464 с.
4. Автоматизированные информационные системы, базы и банки данных. Вводный курс: Учебное пособие. – М.: Гелиос АРВ, 2002. – 368 с., ил.
5. Б.Н. Гайфуллин, И.А. Обухов. Автоматизированные системы управления предприятиями стандарта ERP/MRPII. Производственное издание. – М. «Богородский печатник», 2001, 104 с.
6. Петров В. Н. Информационные системы. – СПб.: Питер, 2002. - 688 с.
7. IEEE Standard Glossary of Software Engineering Terminology/IEEE Std 610.12-1990
8. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.
9. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД "Вильямс", 2002.
10. Алистер Коберн. Современные методы описания функциональных требований к системам. М.: издательство «Лори», 2002. – 263 с.
11. Мацяшек Лешек. Анализ требований и проектирование систем. Разработка информационных :с Диалектика-Вильямс
12. Орлик С., Булуй Ю. Введение в программную инженерию и управление жизненным циклом ПО Программная инженерия. Программные требования. Copyright © Сергей Орлик, 2004-2005. http://www.sorlik.ru/swebok/3-1-software_engineering_requirements.pdf
13. IEEE Guide to the Software Engineering Body of Knowledge (1) - SWEBOK®, 2004. – http://www.swebok.org
14. ГОСТ Р ИСО/МЭК 12207/99. Государственный стандарт РФ. Информационная технология. Процессы жизненного цикла информационных систем. Издание официальное. – М., 1999.
15. Л.Новиков. Введение в Rational Unified Process. http://www.interface.ru/rational/interface/151199/rup/main.htm
16. Белые страницы MSF. http://www.microsoft.com/rus/msdn/msf
17. Analyzing requirements and defining Microsoft .Net solution architectures 2000 г. 491 стр. Microsoft Press.
18. Введение в Rational Unified Process/ Ф. Кратчен – СПб.: Вильямс, 2002. – 240 с.

19. Унифицированный процесс разработки программного обеспечения/ А. Якобсон, Г. Буч, Дж. Рамбо – СПб.: Питер , 2002. – 496 с .
20. IEEE 1362 - Concept of Operations Document
21. IEEE 1233 - Guide for Developing System Requirements Specifications.
22. IEEE Standard 830-1998, «IEEE Recommended Practice for Software Requirements Specifications»

2. Требования и их свойства. Процесс анализа требований

План лекции

Свойства требований
Полнота
Ясность
Корректность и согласованность (непротиворечивость)
Верифицируемость
Необходимость и полезность при эксплуатации
Осуществимость
Трассируемость
Упорядоченность по важности и стабильности.
Наличие количественной метрики
Каких требований не должно быть
Процесс анализа требований
Рабочий поток анализа требований
Кто создаёт и использует требования
Организация работы с требованиями на примере MSF

Ф. Брукс в своём, теперь уже ставшим классическим, эссе[3], следующим образом охарактеризовал роль требований в разработке программного обеспечения. Строжайшее и единственное правило построения систем программного обеспечения (ПО) – решить точно, что же строить. Никакая другая часть концептуальной работы не является такой трудной, как выяснение деталей технических требований, в том числе и взаимодействие с людьми, с механизмами и с иными системами ПО. Никакая другая часть работы так не портит результат, если она выполнена плохо. Ошибки никакого другого этапа работы не исправляются так трудно (конец цитаты).

Наука извлечения и формализации качественных (иногда говорят «хороших», «правильных») требований носит во многом эмпирический характер. Однако, в практике разработки программных систем накопились определённые представления о том, какими свойствами должны обладать требования к программной системе. Это:

- полнота,
- ясность,
- корректность,
- согласованность,
- верифицируемость,
- необходимость,
- полезность при эксплуатации,
- осуществимость,
- модифицируемость,
- трассируемость,
- упорядоченность по важности и стабильности,
- наличие количественной метрики.

Большинство из этих свойств раскрыто в первом разделе стандарта IEEE [1] и широко обсуждается в работах [8,9]. Рассмотрим указанные выше свойства подробнее.

Полнота. Как известно из теории искусственного интеллекта, неполнота – одно из фундаментальных свойств человеческого знания. При создании программных систем нам приходится иметь дело с характеристиками ещё несуществующей системы. Идея о

[3] Brooks, Frederick P. Jr. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. C River, NJ: Prentice Hall PTR.

том, что необходимо сформулировать все требования полностью, т.е. исчерпывающим образом, до начала проектирования, а тем более – реализации системы, изжила себя вместе с так называемым каскадным подходом[4] [2], который поддерживал последовательную модель реализации системы. Спиральный[5] [2] подход, на котором базируется большинство современных методологий, предусматривает поэтапное выделение и детализацию требований на всём протяжении цикла разработки системы.

Тем не менее, требование полноты предъявляется к требованиям, формулируемым к системе. Надо понимать, что данное требование – это скорее тенденция, цель, к которой нужно постараться максимально приблизиться на как можно более ранних стадиях проекта.

Требование полноты можно рассматривать в двух аспектах: полнота отдельного требования и полнота системы требований.

Полнота отдельного требования – свойство, означающее, что текст требования не требует дополнительной детализации, то есть в нём предусмотрены все необходимые нюансы, особенности и детали данного требования.

Полнота системы требований – свойство, означающее, что совокупность артефактов, описывающих требования, исчерпывающим образом описывает всё то, что требуется от разрабатываемой системы.

Ясность (недвусмысленность, определённость, однозначность спецификаций). Каждый из совладельцев[6] разрабатываемой системы обладает своим личным опытом восприятия событий внешнего мира. Слово, произнесённое вслух, вызывает индивидуальные ассоциации в семантическом пространстве каждого отдельного воспринимающего субъекта. То, что является ясным, допустим, для кардиохирурга, совсем необязательно будет таковым для специалиста в области программной инженерии.

Соответственно, требование обладает свойством ясности, если оно сходным образом воспринимается всеми совладельцами системы. На практике ясность требований достигается в том числе и в процессе консультаций, в ходе которых происходит «выравнивание тезаурусов» совладельцев системы. Хорошим подспорьем в этом служит согласованный сторонами глоссарий ключевых понятий предметной области.

К.Вигерс [8] даёт следующий совет по повышению ясности документов: «Пишите документацию просто, кратко и точно, применяя лексику, понятную пользователям».

Ещё одной стороной понятия «ясность требования» является его прослеживаемость (см. также понятие трассируемости ниже по тексту). Требование, которое сформулировано ясно, может быть прослежено, начиная от того документа, где оно сформулировано впервые, вплоть до рабочих спецификаций.

Корректность и согласованность (непротиворечивость). Корректность – одно из важнейших свойств требований. К. Вигерс в [8] вводит понятие корректности требования через точность описания функциональности. В этом смысле корректность в определённой степени конкурирует с полнотой. Но есть и различие: если свойство полноты носит скорее качественный характер: абсолютная полнота представляет недостижимый идеал, к которому можно приближаться, то свойство корректности носит оценочный характер и задаёт дихотомию: каждое из требований либо корректно, либо нет. Кроме того, можно рассуждать о взаимной корректности требований или согласованности (непротиворечивости): если два требования вступают в конфликт, значит – как минимум одно из них некорректно. В иерархии требований (см. материалы лекции 2) можно выделить вертикальную и горизонтальную согласованность. Иными словами, требования

---

[4] Royce, Walker W. Managing the development of large software systems: concepts and techniques. Proc. IEEE WESTCON, Los Angeles, August 1970, pp. 1-9

[5] Boehm, B.W. A spiral model of software development and enhancement. IEEE Computer, 21 (5), 1988, pp. 61-72.

[6] Терминология RUP, см. ниже

не должны противоречить, соответственно, требованиям своего уровня иерархии и требованиям «родительского» уровня. Так, требования пользователей не должны противоречить бизнес-требованиям, а функциональные требования – требованиям пользователя.

Верифицируемость (пригодность к проверке). Признаки (свойства) требований, рассматриваемые в настоящей лекции, нельзя считать независимыми. В математической статистике такие признаки называются кореллируемыми. Так, свойство верифицируемости существенно связано со свойствами ясности и полноты: если требование изложено на языке, понятном и одинаково воспринимаемым участниками процесса создания информационной системы, причём оно является полным, т.е. ни одна из важных для реализации деталей не упущена – значит, это требование можно проверить. При этом в ходе проверки у сторон (принимающей и сдающей работу) не должно возникнуть неразрешимых противоречий в оценках. Методы верификации требований будут рассмотрены в лекции 6. Так как хорошо сформулированные требования составляют основу успешного создания системы – роль верифицируемости трудно переоценить. Требования к системе представляют основу контракта между Заказчиком и Исполнителем и если данные требования нельзя проверить – значит и контракт не имеет никакого смысла, следовательно, успех или неудача проекта будут зависеть только от эмоциональных оценок сторон и их способности договориться, а это – слишком шаткая основа для осуществления работ.

Необходимость и полезность при эксплуатации. Одни из самых субъективных и трудно проверяемых свойств требований.

Возвращаясь к иерархии требований в лекции 2, наиболее бесспорными требованиями следует считать бизнес-требования. Данные требования формулируют первые лица, представляющие Заказчика, и вряд ли кто-нибудь лучше них сможет сказать, каким условиям должна соответствовать создаваемая информационная система, чтобы соответствовать бизнес-целям предприятия. Тем не менее, если у представителя Исполнителя возникают сомнения в необходимости того или иного бизнес-требования, вызванные интуитивными соображениями, либо опытом внедрения информационных систем на аналогичных предприятиях, он должен проявить инициативу и собрать совместное совещание сторон. Аргументы в пользу отсутствия необходимости требования несомненно будут восприняты, особенно если они будут мотивированы в бизнес-терминологии Заказчика и подтверждены выкладками, прогнозирующими соотношение затрат на выполнение требования и ожидаемой от него эффективности.

Необходимость требований пользователя может вытекать из соответствующих бизнес-требований. Кроме того, требования пользователя могут мотивироваться эргономичностью продукта и особенностями функционирования его отдела (подразделения), недостаточно полно раскрытыми на предыдущем уровне иерархии требований.

Большинство функциональных требований вытекают из требований первых двух уровней. Другие функциональные требования могут лежать вне сферы компетенции Заказчика (который, вообще говоря, не обязан быть экспертом в области IT) и их должен сформулировать Исполнитель. Так, например, информационная система в процессе её использования может начать снижать свою производительность из-за больших объёмов накапливаемых данных. Поэтому целесообразно заложить функции архивирования информации, переключения учётных периодов и т.п., необходимость которых следует не из особенностей бизнеса предприятия внедрения, а из общих принципов построения информационных систем.

Более слабой, чем «необходимость» формулировкой обладает свойство «полезность при эксплуатации». Разграничение между данными свойствами можно провести следующим образом. Необходимыми следует считать свойства, без выполнения которых невозможно, либо затруднено выполнение автоматизированных бизнес-функций

пользователей; полезными при эксплуатации следует считать любые свойства, повышающие эргономические качества продукта.

Осуществимость (выполнимость). Является в некоторой степени конкурирующим по введённым выше двум свойствам.

В принципе никто не мешает сформулировать требование, выполнимость которого ограничивается сегодняшним уровнем развития техники и технологии, хотя многое из того, что было невыполнимо десять лет назад, вполне выполнимо сегодня. Можно сформулировать требование, выполнимость которого ограничена научными представлениями о строении Вселенной, например – требование мгновенной передачи информации с земной станции на Марс, хотя и фундаментальные представления иногда меняются, пусть и не так быстро, как развитие IT-технологий.

Возвращаясь с небес на землю, отринем те требования, которые можно признать абсурдными и остановимся на тех, которые выполнимы принципиально. С точки зрения науки управления требованиями, далеко не все из них являются осуществимыми.

Выполнимость требования на практике определяется разумным балансом между ценностью (степенью необходимости и полезности) и потребными ресурсами. Так, если стоимость контракта на разработку информационной системы составляет $10000, а затраты на выполнение нового требования, возникшее в момент, когда проект выполнен наполовину, оценивается в $4000, является ли оно невыполнимым? Скорее всего, да, если Исполнитель докажет Заказчику новизну требования (требование не входило в согласованные спецификации) и сложность его исполнения. Но, если требование является критически важным, необходимым, но выпало из поля зрения при подписании контракта Заказчик готов выделить дополнительно финансирование, а Исполнитель – трудовые ресурсы – значит, требование выполнимо. Таким образом, требование осуществимости в ряде случаев также следует считать субъективным, а критерии его оценки лежат в области договорённостей между Заказчиком и Исполнителем.

Отличной иллюстрацией балансировки между ценностью и выполнимостью требований является так называемый треугольник компромиссов.

Рис. 3-1.

В качестве пояснения к рисунку приведём цитату из «белых страниц», размещённых Microsoft в открытом доступе [4]. Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (рамками). Эти три переменные образуют треугольник, показанный на рис. 3-1. После достижения равновесия в этом треугольнике изменение на любой из его сторон для поддержания баланса требует модификаций на другой (двух других) сторонах и/или на изначально измененной стороне.

Необходимо обеспечить возможность переработки требований, если понадобится, и поддерживать историю изменений для каждого положения. Для этого все они должны быть уникально помечены и обозначены, чтобы вы могли ссылаться на них однозначно. Каждое требование должно быть записано в спецификации только единожды. Иначе вы легко получите несогласованность, изменив только одно положение из двух одинаковых.

Лучше используйте ссылки на первоначальные утверждения, а не дублируйте положения. Модификация спецификации станет гораздо легче, если вы составите содержание документа и указатель. Сохранение спецификации в базе данных коммерческого инструмента управления требованиями сделает их пригодными для повторного использования (конец цитаты).

Трассируемость. Трассируемость требования определяется возможностью отследить связь между ним и другими артефактами информационной системы (документами, моделями, текстами программ и пр.). Отдельная траса представляет собой направленное бинарное отношение, заданное на множестве артефактов ИС, где первый элемент отношения представляет соответствующее требование, а второй – артефакт, зависимый от данного требования. На практике трассировки анализируются при посредстве графовых, либо табличных моделей.

Процесс трассировки позволяет, с одной стороны, выявить уже на стадии проектирования системы проектные артефакты, к которым не ведёт связь ни от одного из артефактов, описывающих требования, с другой – артефакты, описывающие требования, не связанные с проектными артефактами. В первом из случаев целесообразно убедиться в том, что проектный артефакт действительно имеет право на существование, а не является избыточным. Во втором случае необходимо проанализировать полезность выявленных требований: либо эти требования несут недостаточную полезную нагрузку и могут быть игнорированы, либо имеют место ошибки проектирования: пропущены соответствующие артефакты.

Другая цель трассировки – повысить управляемость проектом: при изменении отдельно взятого требования становится понятно – какие из проектных, рабочих и других артефактов подлежат изменению (см. также материалы лекции 6).

Упорядоченность по важности и стабильности. *Приоритет требования* представляет собой количественную оценку степени значимости (важности) требования. Приоритеты требований обычно назначает представитель Заказчика. Разработчик, отталкиваясь от приоритетности требований, управляет процессом реализации информационной системы.

Стабильность требования характеризует прогнозную оценку неизменности требований во времени.

Наличие количественной метрики. Количественные метрики играют важную роль в верификации и аттестации информационных систем. В первую очередь это относится к нефункциональным требованиям, которые, как правило, должны иметь под собой количественную основу (запрос должен отрабатываться не более, чем ___ секунд; средняя наработка на отказ должна составлять не менее, чем ___ часов). Функциональные требования также могут расширяться количественными мерами при помощи так называемых аспектов применимости (см. материал лекции 5).

Каких требований не должно быть

Согласно [9], спецификация требований не должна содержать деталей проектирования или реализации (кроме известных ограничений). Иными словами, требования должны отвечать на вопрос: «что должна делать система», абстрагируясь от вопроса «как она это должна делать». Стремление принимать детальные проектные решения на этапе анализа требований – одно из типичных «ловушек», типичных для неопытных команд разработчиков. Вариантов реализации всегда больше, чем один, а для принятия взвешенного решения нужна максимально более полная информация. Поэтому этапы работы с требованиями, проектирования и реализации планируются поочерёдно, хотя и могут быть частично запараллелены в рамках итерационного подхода к созданию программных систем (см. материалы заключительной лекции).

Рабочий поток анализа требований

Анализ требований – один из основных рабочих потоков (workflow) программной инженерии, наряду, допустим, с такими, как проектирование интерфейса пользователя, либо программирование.

Для его обозначения в англоязычной литературе, как правило, используется понятие «Requirement Process». В отечественной практике, наряду с обобщающим термином «анализ требований», принятым, в частности, в ГОСТ Р ИСО/МЭК 12207-99, встречаются также такие термины, как «поток работ «требования», «работа с требованиями», «определение требований» и т.д., что вносит изрядную путаницу. Для того, чтобы внести некоторую ясность, рассмотрим декомпозицию рабочего потока Requirement Process на составляющие, принятую в SWEBOK, и введём терминологию, которой будем придерживаться на протяжении лекционного курса.

SWEBOK предлагает выделить в Requirement Process следующие основные составляющие:

- Requirements Elicitation (Извлечение требований),
- Requirements Analysis (Анализ требований в узком смысле),
- Requirements Specification (Специфицирование требований),
- Requirements Validation (Проверка требований).

В качестве примера альтернативной декомпозиции потока работ можно рассмотреть взгляд, предложенный в RUP [6]. RUP предлагает выделить в основном потоке анализа требований такие компоненты, как:

- Analyze the Problem (Анализ проблемы),
- Understand Stakeholder Needs (Понимание потребностей совладельцев),
- Define the System (Определение системы),
- Manage the Scope of the System (Управление контекстом системы),
- Refine the System Definition (Уточнение определения системы).

Обобщая указанные выше декомпозиции, а также подходы, описанные в [9,10-12], далее будем придерживаться следующей, более удобной в методическом плане схемой декомпозиции потока работ «Работа с требованиями»:

- Формирование видения;
- Выявление требований;
- Классификация и спецификация требований;
- Расширенный анализ требований (моделирование и прототипирование);
- Документирование требований;
- Проверка требований;
- Управление требованиями;
- Совершенствование процесса работы с требованиями.

Первые 6 работ в лекционном курсе рассматриваются, как компоненты процесса анализа требований.

Для того, чтобы успешно создать автоматизированную информационную систему (или шире, программную систему), необходимо, во-первых, определить компоненты потока работ, которые будут использоваться командой разработчиков и, во-вторых, правильно их организовать. В вопросы организации входит упорядочение работ во времени, интерфейсы между ними, параллелизм, работа с рисками и многое другое.

Найти ответ на первый вопрос может помочь общая классификация задач, работ и операций программной инженерии, представленная в ГОСТ Р ИСО/МЭК 12207-99. Другая, более поздняя по времени классификация, присутствует в SWEBOK. Однако нужно отметить, что данные руководящие документы рассматривают общий случай, а в частном проекте может быть задействован далеко не весь арсенал работ.

Сложнее – с решением второго вопроса. На сегодня существуют и имеют примеры успешного применения десятки и сотни различных методологий (процессов), среди наиболее известных – MSF, RUP, Oracle PJM, XP, FDD, SCRUM, PSP, Crystal, DSDM. Методологии подразделяются на 3 «волны»: каскадные (исторически первые),

прогнозирующие (например, RUP) и «быстрые» (agile), вошедшие в широкую практику на рубеже тысячелетий [8].

Описания методологий существенно разняться объёмом (от десятков до тысяч страниц текста), наборами базовых работ и рабочих квалификаций, акцентами (работа с моделями, управление рисками, построение команды и пр.). Но авторы их описаний обычно сходятся в одном: лучшая из методологий, которой нужно следовать, чтобы добиться успеха – именно та, которую предлагает (описывает, рекламирует) автор. Редким исключением являются работы А. Коберна, автора группы методологий Crystal (см., в частности, [8]), где он предлагает брать за основу не «самый лучший» из процессов, а тот, который, во-первых, наилучшим образом соответствует проектной задаче, а во вторых – команде, которая будет его реализовывать. В [8] вводится несколько метрик, позволяющих частично формализовать процесс подбора методологии.

Почему нужно анализировать требования?

Из сказанного выше следует, что не все работы и операции, известные в программной инженерии, используются в той или иной методологии и, тем более, конкретном проекте. Возникает вопрос: является ли рабочий поток АТ необходимым в цепочке рабочих потоков создания информационной системы, стоит ли тратить на него время? Каков требуемый объём результатов, ожидаемых от АТ?

Со всей очевидностью можно утверждать: да, АТ, как этап разработки ИС, невозможно пропустить: этот этап закладывает фундамент всего процесса проектирования и реализации системы. Степень проработки АТ может быть различной: от совершенно неформальной записки, представленной на одной странице, до развёрнутой системы документов, моделей и прототипов, построенной в соответствии с принципами одной из прогнозирующих методологий, например, RUP. Она зависит от следующих основных факторов: размеров проекта, величины имеющихся ресурсов и степени рисков. Невысокая глубина проработки приемлема для небольших проектов, характеризующихся небольшим ресурсом и невысокими рисками. Хорошо проработанные требования позволяют:

- выработать общее понимание между Заказчиком и Разработчиком;
- определить рамки проекта;
- более точно определить финансовые и временные характеристики проекта;
- обезопасить Заказчика от риска получить продукт, в котором он не сможет работать,
- обезопасить Разработчика от риска попасть в ситуацию «неконтролируемого размытия границ», которое может привести к непредвиденным затратам ресурсов сверх начальных ожиданий.

Анализ требований – это процесс (бизнес-процесс) и, следовательно, к нему подходят методы и средства процессного подхода к управлению (см., например, [6]).

Один из ключевых вопросов, позволяющих оценить результативность процесса – что является *выходом* (результатом) процесса. В чём результат АТ? Результатом рабочего потока «анализ требований» является набор артефактов. Это могут быть текстовые документы, модели UML, либо других языков моделирования, прототипы программного обеспечения.

Другой важный вопрос – какие *цели* преследует процесс.

RUP предлагает следующие цели для потока работ АТ:

- добиться одинакового понимания с заказчиком и пользователями о том, что должна делать система;
- дать разработчикам наилучшее понимание требований к системе;
- определить границы системы;
- определить интерфейс пользователя и системы.

Кто создаёт и использует требования

Как и кем используются требования?

Специалист по АТ – постановка задачи, определение рамок проекта,

Представитель заказчика – постановка задачи, определение рамок проекта, контроль работы исполнителя, приёмка результатов работы.

Архитектор системы – разработка архитектуры, проектирование подсистем

Программист – разработка программного кода.

Тестировщик – составление тест-плана, тестовых сценариев.

Менеджер проекта – планирование и контроль исполнения работ.

В рамках курса лекций для всех упомянутых выше лиц будем использовать обобщающий термин «Совладельцы (заинтересованные стороны)» (stakeholders). Совладельцами, вслед за разработчиками RUP и MSF (см., например, [9,13]), будем называть всех участников проекта создания программной системы, прямо или косвенно заинтересованных в его успехе. Авторы большинства современных методологий разработки программных систем сходятся том, что в группе совладельцев ключевую роль играют две группы представителей Заказчика – те, кто ставит стратегические цели и выделяет финансирование и те, кто будет непосредственно использовать разработанный продукт. Причём, в отличие от каскадных методов, где Заказчик подключался в начальной фазе – составлении технического задания и конечной – приёмке готовой работы, в современных методологиях Заказчик, действительно заинтересованный в успехе проекта автоматизации, должен участвовать в нём **непрерывно**.

Организация работы с требованиями на примере MSF

В MSF для обозначения роли участников команды софтверного проекта используется понятие ролевых кластеров [4].

MSF основан на постулате о шести качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают модель проектной группы. В то время как за успех проекта ответственна вся команда, каждый из ее ролевых кластеров, определяемых моделью, ассоциирован с одной из упомянутых шести целей и работает над ее достижением.

Шесть ролевых кластеров модели проектной группы – это “Управление продуктом” (product management), “Управление программой” (program management), “Разработка” (development), “Тестирование” (test), “Удовлетворение потребителя” (user experience) и “Управление выпуском” (release management). Они ответственны за различные области компетенции (functional areas) и связанные с ними цели и задачи.

MSF организован на базе комбинации каскадной и спиральной моделей. Отдельная стадия работы содержит в себе 5 фаз:

- Envisioning (выработка концепции),
- Planning (планирование),
- Developing (разработка),
- Stabilizing (стабилизация),
- Deploying (внедрение).

В фазе выработки концепции работа с требованиями наиболее интенсивна (см. табл. 1).

Табл. 1.

Ролевой кластер	Фокус
Управление продуктом	Общие цели проекта; выявление нужд и требований заказчика; документ общего описания и рамок проекта.
Управление программой	Цели дизайна; концепция решения; структура проекта.
Разработка	Прототипирование; анализ технологических возможностей; анализ осуществимости.
Удовлетворение потребителя	Необходимые эксплуатационные характеристики решения и их влияние на его разработку.
Тестирование	Стратегии тестирования; критерии приемлемости, их влияние на разработку решения.
Управление выпуском	Требования внедрения и их влияние на разработку решения; требования сопровождения.

Как видно из таблицы, все 6 кластеров работают со своими группами требований.

Продолжается плотная работа с требованиями и на следующей фазе – фазе планирования, см. табл. 2.

Табл. 2.

Ролевой кластер	Фокус
Управление продуктом	Анализ бизнес-требований
Управление программой	Функциональная спецификация
Удовлетворение потребителя	Сценарии/примеры использования, пользовательские требования, требования локализации и общедоступности (accessibility).
Тестирование	Требования тестирования.
Управление выпуском	Эксплуатационные требования.

В фазах разработки и внедрения работа с требованиями сосредотачивается в кластерах управления продуктом и программой, см., соответственно, табл. 3,4.

Табл. 3.

Ролевой кластер	Фокус
Управление продуктом	Ожидания заказчика.
Управление программой	Управление функциональной спецификацией.

Табл. 4.

Ролевой кластер	Фокус
Управление продуктом	Получение отзывов и оценок заказчика; акт о приеме выполненной работы.
Управление программой	Сопоставление рамок проекта с поставленным решением; управление стабилизацией.

Литература к лекции

1. IEEE Standard 830-1998, «IEEE Recommended Practice for Software Requirements Specifications»
2. Петров В. Н. Информационные системы. – СПб.: Питер, 2002. - 688 с.
3. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.
4. Microsoft Solutions Framework. Модель процессов MSF, версия 3.1 http://www.microsoft.com/Rus/Download.aspx?file=/Msdn/Msf/MSF_process_model_rus.doc
5. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД “Вильямс”, 2002.
6. Введение в Rational Unified Process/ Ф. Кратчен – СПб.: Вильямс, 2002. – 240 с.
7. Каменова, Громов. Моделирование бизнеса. Методология ARIS. — М.: Весть-МетаТехнология, 2001.
8. Коберн А. Быстрая разработка программного обеспечения. – М.: Лори, 2002. 314 с.
9. Белые страницы MSF. http://www.microsoft.com/rus/msf
10. Орлик С., Булуй Ю. Введение в программную инженерию и управление жизненным циклом ПО Программная инженерия. Программные требования. Copyright © Сергей Орлик, 2004-2005. http://www.sorlik.ru/swebok/3-1-software_engineering_requirements.pdf
11. Мацяшек Лешек, А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер. с англ. - М.: Издательский дом "Вильямс", 2002. - 432 с.: ил. - Парал. тит. Англ.
12. Брауде Э. Технологии разработки программного обеспечения. – СПб: Питер, 2004. – 655 с.: ил.
13. Унифицированный процесс разработки программного обеспечения/ А. Якобсон, Г. Буч, Дж. Рамбо – СПб.: Питер , 2002. – 496 с.

3. Контекст задачи анализа требований. Выявление требований.

План лекции

Свойства требований
Анализ требований, бизнес-анализ, анализ проблемной области
Методологии бизнес-анализа
Требования и архитектура АИС
Анализ требований и другие рабочие потоки программной инженерии
Анализ требований, бизнес-анализ, анализ проблемной области
Источники требований
Стратегии выявления требований
Интервью
Анкетирование
Наблюдение
Самостоятельное описание требований
Совместные семинары
Прототипирование

В настоящее время существуют уже сотни методик, методологий, процессов, стандартов, регламентирующих те или иные детали выбора и комплексирования потоков работ при разработке автоматизированных информационных систем. То, что в АТ стоит в начале цепочки работ и что её результаты во многом определяют успех проекта мало у кого вызывает сомнения. Другое дело – работы, связанные с бизнес-анализом и бизнес-моделированием. Их роль не столь очевидна и принимается далеко не всеми методологиями. Итак, стоит ли собирать информацию о предприятии, для которого разрабатывается (выбирается) АИС в виде бизнес-моделей или стоит пропустить этот этап и сразу формировать артефакты АТ?

Авторы [1], «отцы-основатели» RUP и UML, в этом вопросе дают определённую свободу: можно создавать бизнес-модели при помощи соответствующих расширений UML и рекомендаций RUP, а можно ограничиться выработкой глоссария объектов предметной области. Как и в вопросе выбора глубины проработки артефактов АТ, вопрос – проводить или не проводить бизнес-анализ (или, точнее говоря, анализ проблемной области), решается в зависимости от конкретной задачи.

Роль глоссария при АТ. Несколько утрируя, можно сказать, что Заказчик и Разработчик всегда говорят на разных языках. Общее понимание вырабатывается с трудом, этот процесс занимает время, но важность его трудно переоценить: ведь успешная реализация проекта в области и внедрения АИС во многом зависит от того, удастся ли выработать и документировать их общее представление о предмете разработки. Если же Разработчик идёт ещё дальше и вникает в особенности ведения дел на предприятии Заказчика – он, во-первых, сможет добиться лучшего понимания требований к АИС и, во-вторых, участвовать наряду с Заказчиком в формулировке этих требований, анализе пропущенных требований и пр. Глоссарий (подробнее см. в. лекции 4) можно рассматривать, как документ, удостоверяющий общее понимание основной терминологии Заказчиком и Разработчиком.

Задачу анализа бизнес-процессов (деловое моделирование), столь популярную в последние десятилетия ввиду устойчивой конъюнктуры, следует рассматривать, как часть более общей задачи, анализа проблемной области. Работы, посвящённые анализу проблемной области, появились в отечественной литературе в середине прошлого века; данная тематика неразрывно связано с задачным подходом и инженерией экспертных систем. Применимы ли методы, принятые при построении интеллектуальных систем для такой «более приземлённой» задачи, как задача построения АИС – безусловно, да. Так,

стратегии извлечения знаний, рассмотренные в [2], во многом пересекаются с рекомендациями по работе аналитика [3], методы решения задачи путём редукции на подзадачи и поиска в пространстве состояний нашли своё отражение во множестве методик бизнес-анализа, анализа и синтеза программных систем и этот список можно продолжать. Другой вопрос – насколько результативно применение тех или иных моделей и методов при описании организационных систем.

Ключ к решению этого вопроса лежит в следующем: вначале надо определить цели и задачи самого бизнес-анализа, как этапа построения КИС.

С позиций моделирования, анализ требований (АТ) и анализ проблемной области (АПО) – принципиально разные процессы.

АПО преследует классические цели создания модели: налицо объект (автоматизируемое предприятие или организационная система, ОС) и задача аналитика – отразить этот объект в создаваемой модели с требуемой степенью точности (рис. 5-1).

Анализ требований, напротив, направлен на моделирование воображаемого, ещё не существующего объекта (АИС) (рис. 2). Т.е. сначала создаётся модель, а затем, на её основании, синтезируется объект.

Для того, чтобы прояснить связь между этими процессами, необходимо заметить, что создаваемая АИС также является моделью, по отношении к ОС. Таким образом, создавая документ АТ, мы тем самым порождаем как бы «модель второго порядка», т.к. документ АТ является ничем иным, как моделью модели ОС. Не обладая моделью АПО, мы, конечно, можем создать модель АТ. Но при этом мы рискуем тем, что при синтезе оригинала модели (т.е. АИС), не обладая знаниями об ОС, мы можем попасть в ситуацию рассогласования: результирующая АИС не будет ингерентна (согласована с) ОС и, тем самым, не станет жизнеспособной.

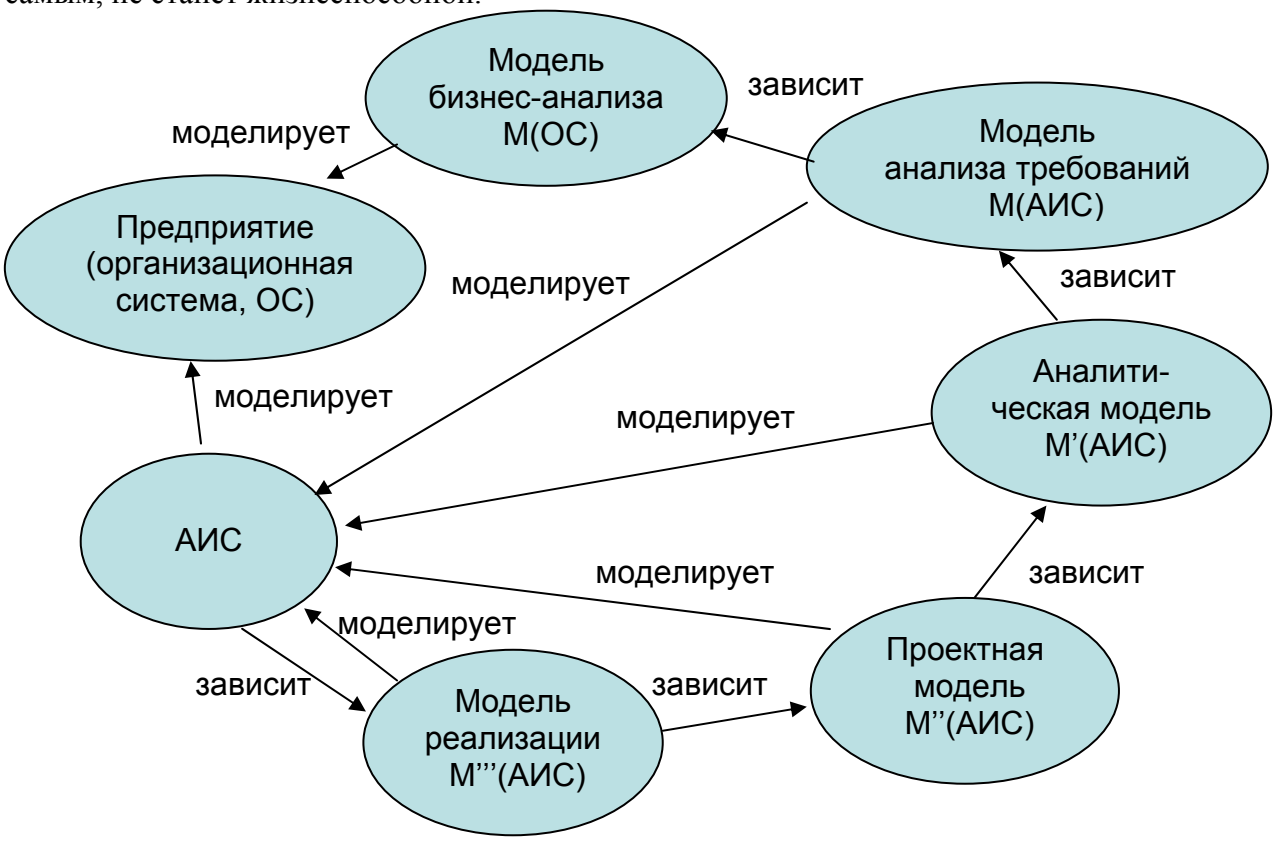


Рис. 5-1.

Следует ли из этого, что этап АПО является необходимым звеном создания КИС? Нет, не всегда. Здесь уместно обратиться к классификации задач и методологий А. Коберна [8]. Кроме того, это зависит от состава третьей компоненты «треугольника моделирования» – моделирующего субъекта, в нашем случае – коллектива Разработчика.

Если моделирующий субъект обладает неявными знаниями об ОС в достаточном объёме – значит, АПО можно исключить. На практике это возможно в следующих случаях: а) Разработчик является частью (структурным подразделением, дочерним предприятием и т.д.) ОС, в коллектив Разработчика входят эксперты, хорошо знающие предметную область; б) Заказчик наравне и Разработчиком участвует в создании документа АТ и разделяет с ним ответственность за принятие решений. Это – путь «agile методологий» (см. материалы заключительной лекции).

Рассмотрим теперь обобщённую «формулу» создания АИС.
ОС → М(ОС) → М(АИС) →М'(АИС) → М''(АИС) → М'''(АИС) → АИС
(рис. 5-1).

Анализ организационной системы позволяет создать модель её модель М(ОС). Это – модель бизнес-анализа (проблемной области).

Анализируя модель проблемной области, в ней можно вычленить, с одной стороны, задачи и функции, реализуемые внутри ОС и функции коммуникации ОС и среды, с другой – устройство предметной области (в начале – на уровне концептуальной модели), с третьей – требования к информации и её обработке. Выделив среди функций те, которые подлежат автоматизации, мы получаем основу для выявления функциональных требований к системе. Остальная, собранная на этапе АПО, информация служит для поиска нефункциональных требований. В результате получаем модель АТ, как первое приближение модели АИС, М(АИС).

Затем, путём углублённого анализа и проектирования, формируются, соответственно, аналитическая модель М'(АИС), проектная модель М''(АИС) и модель реализации М'''(АИС).

Модель уровня реализации позволяет синтезировать собственно АИС, как совокупность программных, информационных, организационных и др. артефактов.

АИС в свою очередь представляет собой модель организационной системы М'(ОС), замыкая цикл моделирования.

Методологии бизнес-анализа

Методологии бизнес анализа можно разделить на три категории по типам моделей:

- модели, преследующие цель анализа и улучшения организационной системы (например, SWOT , VCM, BPR, CPI/TQM/ISO9000, BSC),
- модели общего назначения, такие, как SADT, DFD, IDEF1, IDEF3, IDEF5 и другие.
- модели, специально разработанные для использования при автоматизации (например, ISA, BSP, ARIS, RUP).

Наиболее развитая модель описания проблемной области предлагается в методологии ARIS.

Архитектура ARIS [5] выделяет в организации следующие подсистемы.

- Организационная. Определяет структуру организации — иерархию подразделений, должностей и конкретных лиц, многообразие связей между ними, а также территориальную привязку структурных подразделений.
- Функциональная. Определяет функции, выполняемые в организации.
- Подсистемы входов/выходов. Определяют потоки используемых и производимых продуктов и услуг.
- Информационная (подсистема данных). Описывает получение, распространение и доступ к информации (данным).
- Подсистема процессов управления. Определяет логическую последовательность выполнения функций посредством событий и сообщений. Можно сказать, что подсистема управления — это совокупность разнесенных во времени сообщений разного рода.

- Подсистема целей организации. Описывает иерархию целей, достигаемых в ходе выполнения того или иного процесса.
- Подсистема средств производства. Описывает жизненный цикл основных и вспомогательных средств производства.
- Подсистема человеческих ресурсов. Описывает прием на работу, обучение и продвижение по службе персонала организации.
- Подсистема расположения организационных структур. Описывает территориальное расположение организационных единиц (конец цитаты).

Данное разделение является в определённой мере условным; выделенные «подсистемы» не являются подсистемами в смысле системного анализа, т.к. взаимопроникают и пересекаются. Они представляют скорее совокупность предметов исследования, разных взглядов на исследуемый объект.

Слушателю курса предлагается самостоятельно проанализировать, какие группы и категории требований к системе позволяет прояснить та или иная компонента принятой в ARIS структуризации объекта исследования.

Требования и архитектура АИС

Говоря об архитектуре АИС, обычно подчёркивают деление на аппаратные, программные, информационные, организационные компоненты, их связность и детализацию.

Метафора архитектуры RUP описывается в виде 4+1 представлений: логическое, представление процессов, представление реализации и физическое представление связываются между собой представлением вариантов использования (use case), которое играет центральную роль в выработке архитектуры системы (рис. 5-2).

Требования первичны по отношению к архитектуре. Но это не значит, что требования и архитектура должны разрабатываться последовательно.

Рис. 5-2

Напротив, эти процессы взаимоувязаны и должны быть существенно запараллелены. Как только собран минимальный набор ключевых требований, дающих понимание о том, что нужно делать – должна быть найдена архитектура, объясняющая, как это может быть реализовано. В крупных, ответственных проектах обычно рассматривается несколько альтернативных архитектур, их достоинства и недостатки применительно к исходным требованиям.

В процессе работы с требованиями они детализируется, детализируется и архитектура. В случае множественности альтернативных архитектур на определённом уровне детализации необходимо остановиться на одной, чтобы не распылять ресурсы. Но природа требований такова, что, помимо детализации они ещё и изменяются. С изменением требований изменяются и детали архитектуры. Устойчивость архитектуры проявляется в незначительных её изменениях при добавлении, детализации и изменении

требований. Если наступил момент, при котором появление новой информации о требованиях перестала оказывать влияние на архитектуру – значит, архитектура стабилизировалась.

Это – нормальный, естественный путь развития требований и архитектуры. Но что делать, если требования изменились настолько, что архитектура перестала им соответствовать? Причин тому могут быть разные, например: неопытная архитектурная группа не проявила достаточно дальновидности; группа по сбору требований пропустила на ранних стадиях критичные, архитектурно значимые требования; в бизнес-сфере Заказчика произошли большие перемены, вызвавшие коренное изменение требований к системе. Следствия также могут быть различными: договорённость об увеличении сроков и сумм по контракту; исправление ситуации за счёт Разработчика; разрыв контракта.

Альтернативный выход предлагается в методологии XP: архитектура – не догма, а всего лишь метафора. Если требования вошли вразрез с существующей архитектурой – значит, архитектуру нужно просто изменить. Следует понимать, что путь и рецепты XP при кажущейся простоте ориентированы далеко не на любой коллектив. Команда XP состоит их профессионалов, имеющих позитивный опыт работы в этой методологии.

Анализ требований и другие рабочие потоки программной инженерии

Рассмотрим краткий обзор рабочих потоков RUP и их связь с потоком работ АТ (рис. 5-3).

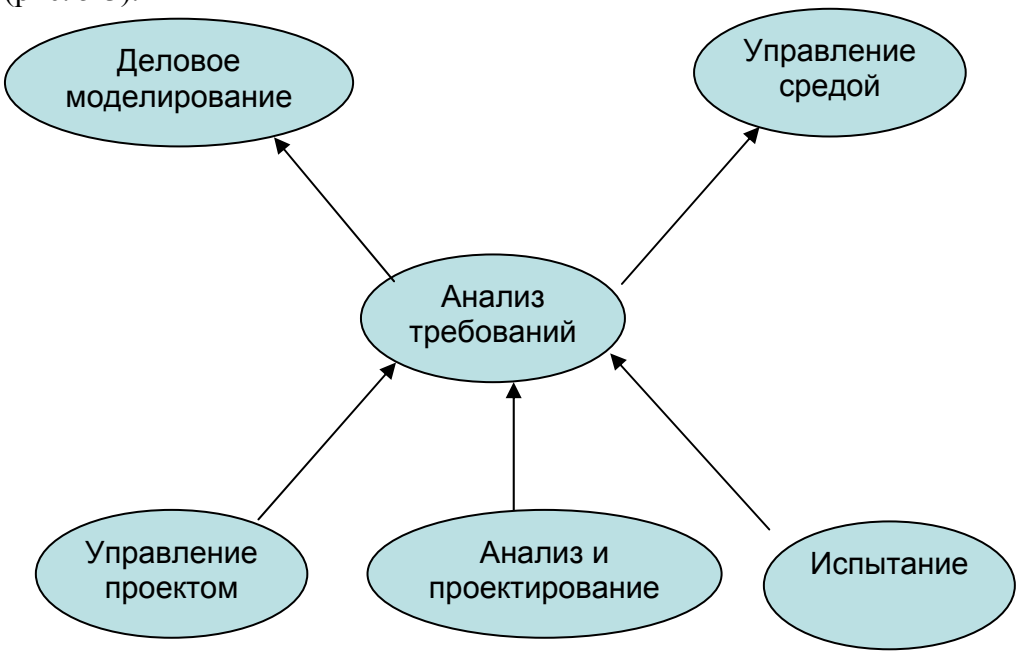


Рис. 5-3

Поток работ «деловое моделирование» служит основой для анализа и формирования требований к АИС, позволяет избежать ошибок.

Поток работ «управление средой» предоставляет исходную информацию для рабочей группы АТ, регламентирующую форматы оформления, CASE-средства, регламенты работы.

Поток работ «управление» основывается на спецификации требований. Стратегическое и тактическое планирование, формирование промежуточных вех (ожидаемых результатов) тесно увязано с требованиями к системе.

Поток работ «анализ и проектирование» осуществляется на основе исходных данных, предоставленных АТ. В определённой мере эти потоки работ проводятся параллельно. При обнаружении проблем, связанных с требованиями, возникает обратная связь от этого потока работ к потоку работ АТ.

Поток работ «испытание» во многом базируется на модели требований и дополнительных спецификациях, регламентирующих процесс тестирования (тестовые сценарии и пр.).

Для потока работ «реализация» связь с требованиями не указана. Между тем автор считает, что требования должны анализироваться и учитываться во ВСЕХ рабочих потоках проекта, даже если это формально не предусмотрено выбранным группой процессом. Людям свойственно ошибаться и ошибки, совершённые на ранних стадиях проекта, при движении от этапа к этапу нарастают, как снежный ком. Поэтому любому участнику команды, заинтересованному в успехе проекта, нелишне заглянуть в спецификацию требований и убедиться в том, что та работа, которая ему поручена, соответствует тому или иному требованию. Это позволяет организовать обратную связь, позволяющую отследить ошибки в спецификациях. Многие проекты зашли в тупик именно из-за оторванности группы, отвечающей за реализацию от группы сбора и анализа требований.

3. Контекст задачи анализа требований. Выявление требований.....24
План лекции 24
Методологии бизнес-анализа 26
Требования и архитектура АИС.....27
Анализ требований и другие рабочие потоки программной инженерии.....28
Источники требований.....29
Стратегии выявления требований.....30
Интервью 30
Анкетирование.....32
Наблюдение.....32
Самостоятельное описание требований 32
Совместные семинары.....33
Прототипирование.....33
Литература к лекции 34

Источники требований.

Основным источником требований к информационной системе, безусловно, являются соображения, высказанные представителями Заказчика. В соответствие с иерархической моделью требований данная информация структурируется как минимум на 2 уровня: бизнес-требования и требования пользователей. Проблема состоит в том, что требования формулируются к создаваемой, ещё не существующей системе, т.е. по сути решается начальная подзадача задачи проектирования АИС, а представители Заказчика далеко не всегда бывают компетентны в данном вопросе. Поэтому, наряду с требованиями, высказанными Заказчиком, целесообразно собирать и требования от других совладельцев системы: сотрудников аналитической группы исполнителя, внешних экспертов и т.д.

Результирующий, часто достаточно сырой материал рассматривается, как документ «Требования совладельцев»[7]. На требования совладельцев обычно не накладывается никаких специальных ограничений.

Продолжая рассуждения, начатые в предыдущей лекции, модель создаваемой информационной системы в определённой мере должна отражать модель ОС.

Поэтому другим важным источником информации, помимо выявления требований, являются артефакты, описывающие предметную область. Это могут быть документы с описанием бизнес-процессов предприятия, выполненные консалтинговым агентством, либо просто документы (должностные инструкции, распоряжения, своды бизнес-правил),

[7] Терминология RUP

принятые на предприятии. Одной из немногих методологий, в которых специально выделяются рабочий поток делового моделирования, является Rational Unified Process.

Ещё одна альтернатива, используемая при выявлении требований – так называемые «лучшие практики», широко используемые в настоящее время в бизнес-консалтинге и при внедрении корпоративных информационных систем. Лучшие практики представляют собой описания моделей деятельности успешных компаний отрасли, используемые длительное время в сотнях и тысячах компаний по всему миру.

Подытоживая сказанное, отметим, что основными источниками, образующими «вход» процесса выявления требований, являются требования, высказанные совладельцами, как таковые или (и) артефакты, описывающие объект исследования. Однако, это – достаточно упрощённый взгляд: чтобы данные поступили «на вход», аналитики требований должны проделать немалую работу, связанную с подбором респондентов и информационных материалов, организацией интервью и т.д.

Стратегии выявления требований

Интервью

Ключевой стратегией выявления требований было и остаётся интервью с экспертами.

В ставшей уже классической, но ничуть не утерявшей актуальность монографии Д.Марко [3] в процессе проведения интервью предлагается выделить три подчинённых процесса: подготовку, проведение интервью (опроса) и завершение. Ниже приводится краткий обзор рекомендаций Д.Марко с акцентом на выявление требований (в монографии даны рекомендации по интервьюированию с целью формирования модели объекта исследования).

1. Подготовка

Подготовка позволяет спланировать процесс опроса и выработать стратегию управления этим процессом. Важность подготовительного этапа вырастает, если респондент является «дефицитным» полезным ресурсом, например – президентом крупной компании.

При подготовке Д.Марко рекомендует следующие шаги:

- выберите нужного собеседника;
- договоритесь о встрече;
- установите предварительную программу встречи;
- изучите сопутствующую информацию;
- согласуйте свои действия с группой проектирования[8].

При выборе собеседника для целей сбора требований определяющими являются две вещи:

- Он действительно является экспертом по данному вопросу;
- Его мнение действительно является ценным при формировании целевого набора требований[9].

Важно заранее оговорить цель встречи и ограничить беседу в пределах часа или менее. Практика показывает, что активное общение в процессе интервью, как правило,

---

[8] В нашем случае – группой аналитиков требований

[9] На практике возможны ситуации, когда требование, сформулированное одним из представителей Заказчика, не подтверждается другим представителем, имеющим большие властные полномочия. Надо отчётливо понимать, что каждое требование в конечном итоге транслируется, с одной стороны, в компоненту информационной системы, а с другой – может быть выражено в определённом количестве денежных знаков, которые Заказчик должен будет выплатить Исполнителю по приёмке работы. Поэтому право формулировать требования и область компетенции того или иного эксперта должны быть формально оговорены внутренним документом Заказчика, с которым следует ознакомится до начала проведения интервью.

ограничивается часом. Если этого времени недостаточно, можно спланировать несколько встреч.

Полезными приёмами являются формирование программы беседы и ознакомление с ней респондента, подробное планирование беседы вплоть до записи подготовленных вопросов. Подготовленное таким образом интервью называют *структурированным* [10]. В дополнение к так построенному интервью автор [10] предлагает проводить *неструктурированное интервью,* «представляющее собой неформальную встречу, которой не свойственны заготовленные впрок вопросы или заранее поставленные цели». Цель такого интервью – пробудить респондента к креативу в области, в которой интервьюер недостаточно хорошо ориентируется.

2. Проведение опроса

Ниже приведёна цитата из [3], с некоторыми сокращениями и исключениям материала, не относящегося к АТ.

В проведении опроса самое важное – правильно организовать и поддерживать поток информации от эксперта к вам. Рекомендуется потратить время на обдумывание верного начала опроса, при сборе информации по возможности использовать записи, заканчивать разговор плавно. Обсудим подробнее каждый из этих пунктов.

Начиная разговор, не забудьте представиться и сформулировать цель встречи. Это поможет избежать недоразумений и даст беседе правильное направление. Кроме того, обговорите возможность ведения записей.

Затем сформулируйте первый вопрос. Помните, что первый вопрос часто задает тон всему разговору, поэтому хорошо продумайте его.

Собирайте информацию, делая записи обо всем (о специальных терминах, взаимосвязях между частями системы и т.п.) и ограничивая время беседы. Запишите SADT-функции и данные, попытайтесь набросать диаграмму. Поддерживайте поток информации, задавая вопросы, которые уточняют и подтверждают ответы.

Прежде всего, не возражайте.

Никогда не задавайте наводящих вопросов или вопросов с короткими ответами "да" или "нет". Вместо этого записывайте то, что вам говорят, и просите подвести итог или дать пояснения.

Вы получите от опроса больше, если вы дадите эксперту возможность говорить то, что он хочет сказать, а не то, что вы хотите услышать (конец цитаты).

3. Завершение

Следите за возникновением следующих ситуаций [3]:

- вы уже получили достаточно информации;
- вы получаете большой объем неподходящей информации;
- обилие информации вас подавляет;
- эксперт начинает уставать;
- у вас с экспертом часто возникают конфликты.

Любая из этих причин - достаточное основание для завершения беседы.

Когда вы считаете нужным закончить опрос, завершайте беседу плавно. Кратко подытожьте основные пункты и сделайте обзор полученных сведений, которые могут быть опущены или неверно истолкованы. Договоритесь о времени следующей встречи, если она нужна, и получите рекомендации для ближайших опросов. Поставьте эксперта в известность, когда и как вы собираетесь использовать полученную информацию и когда вы пришлете ему материал на рецензирование.

Всегда оформляйте материалы опроса сразу же после встречи с экспертом. В этом случае немедленно возникает обратная связь, и вы минимизируете возможность потери важной информации.

Что нужно помнить при опросе

Следующие рекомендации помогают поддерживать непрерывность потока и достоверность информации, поступающей от эксперта [3]:

- делайте паузы, пока эксперт думает. Дайте эксперту возможность решать, что сказать дальше. Никогда не перебивайте, подсказывая ответ или задавая другой вопрос;
- старайтесь не задавать наводящих вопросов, вопросов-подсказок, вопросов, содержащих ответ, потому что это не позволяет эксперту делиться своими знаниями. Старайтесь не задавать контрольных вопросов, так как это прерывает поток информации;
- делайте записи, чтобы сосредоточиться на предмете разговора и чтобы подготовиться к следующему вопросу, но не становитесь стенографом, иначе вы можете потерять контроль над опросом (конец цитаты).

Анкетирование

Анкетирование – самый малозатратный для аналитика способ извлечения информации, он же – и наименее эффективный. Обычно применяется как дополнение к другим стратегиям выявления требований.

Недостатки анкетирования очевидны: респонденты часто бывают неспособны, либо слабо мотивированы в том, чтобы хорошо и информативно заполнить анкету. Велика вероятность получить неполную или вовсе ложную информацию. Преимущество – в том, что подготовка и анализ анкет требуют небольшой ресурс.

Л.Мацяшек [10] рекомендует формулировать в анкетах *вопросы с замкнутым циклом ответов* в одной из следующих трёх форм.

Многоальтернативные вопросы. Эта форма анкеты известна всем, кто когда либо проходил тестирование; может расширяться комментариями респондента в свободной форме.

Рейтинговые вопросы. Представляют предопределённый набор ответов на сформулированные вопросы. Используются такие значения, как «абсолютно согласен», «согласен», «отношусь нейтрально», «не согласен», «абсолютно не согласен», «не знаю».

Вопросы с *ранжированием*. Предусматривает ранжирование (упорядочивание) ответов путём присваивания им порядковых номеров, процентных значений и т.п.

Наблюдение

Наблюдение за работой моделируемой организационной системы - полезная стратегия получения информации (хотя, строго говоря, по результатам наблюдения можно получить модель ОС, а не модель АТ).

Различают *пассивное* и *активное* наблюдение. При активном наблюдении аналитик работает, как участник команды, что позволяет улучшить понимание процессов.

Через наблюдение, а возможно, и участие аналитики получают информацию о происходящих день за днем операциях из первых рук. Во время наблюдения за работой системы часто возникают вопросы, которые никогда бы не появились, если бы аналитик только читал документы или разговаривал с экспертами.

Недостатком этой стратегии является то, что наблюдатель, как и всякий «измерительный прибор», вносит помехи в результаты измерений: сотрудники организации, находясь «под колпаком» могут начать вести себя принципиально по другому, чем обычно.

Самостоятельное описание требований

Документы – хороший источник информации, потому что они чаще всего доступны и их можно "опрашивать" в удобном для себя темпе. Чтение документов - прекрасный способ получить первоначальное представление о системе и сформулировать вопросы к экспертам.

Если опытный аналитик уже исследовал большое число систем такого же типа, что и на предприятии внедрения, он обладает фундаментальными знаниями в

соответствующей предметной области, относительно определенного класса систем. Авторы методологии SADT рекомендуют проводить *самоопрос* с тем, чтобы получить максимальную пользу от своих знаний.

По результатам анализа документов и собственных знаний аналитик может составить *описание* требований и предложить его представителям Заказчика в качестве информации к размышлению, либо – основы для формирования технического задания.

Недостаток этой стратегии – опасность пропуска знаний, специфичных для объекта исследования (в случае самоопроса), либо – неформализованных знаний, эмпирических правил и процедур, широко используемых на практике, но не вошедших в документы.

Совместные семинары

Помимо классического интервью «тет а тет», существует значительное количество методик, предполагающих широкое участие представителей Заказчика и Исполнителя.

Правила *мозгового штурма* предполагают полную раскрепощённость и свободу мнений, даже самых вычурных и на первый взгляд «бредовых». Первое правило мозгового штурма – «полный запрет на любую критику». Всякое высказанное мнение представляет ценность, а полное отсутствие запретов позволяет полноценным образом подключить творческую фантазию.

Затем, на втором этапе, все высказанные мнения тщательным образом обсуждаются, заведомо неприемлемые варианты отсеиваются, формируются коллективные предложения.

Правила *JAD-метода*, считающегося одним из современных способов извлечения требований, были впервые сформулированы в конце 1970-х годов компанией IBM. Участники JAD-совещания:

Ведущий – специалист в области межличностных коммуникаций. Должен ориентироваться в предметной области, но не обязательно хорошо ориентироваться в проблемах IT.

Секретарь – стенографист встречи. Фиксирует её результаты на компьютере. Возможно применение CASE-средств.

Заказчики – пользователи или руководители, основные участники, формирующие, обсуждающие требования и принимающие решения.

Разработчики – аналитики и другие участники проектной команды. Работают в большей части в пассивном режиме с целью наилучшего понимания проблемной области.

Совместные семинары, сохраняя все преимущества режима интервью, привносят дополнительные бонусы: работа в группе более продуктивна, группы быстрее обучаются, более склонны к квалифицированным заключениям, позволяют исключить многие ошибки.

Эта стратегия, очевидно, одна из самых затратных, однако она окупается за счёт меньшего количества ошибок и отказе от формализации в пользу живого общения, выработке общего языка и пр. Некоторые методологии (например, XP) зиждутся на постоянном тесном контакте между Заказчиком и Исполнителем и, если такой возможности нет – XP-проект просто не сможет состояться.

“Разъясняющие встречи” [12] или «запланированный мозговой штурм» – термин, пришедший из общей практики менеджмента и базирующийся на идеях сотрудничества заинтересованных лиц для совместного анализа путей решения проблем, определения и предупреждения рисков и т.п.

Как и проведение интервью, организация семинара требует соблюдения правил, с которыми можно познакомиться в [10,8].

Прототипирование

Прототипирование – ключевая стратегия выявления требований в большинстве современных методологий (подробнее см. в лекции 5). Программный прототип –

«зеркало», в котором видно отражение того, как понял Исполнитель требования Заказчика. Процесс выявления требований путём прототипирования тем более интенсивен, чем это зеркало кривее. Документальный способ выявления требований всегда уступает живому общению. Анализ того, что сделано в виде интерфейсов пользователя даёт ещё больший эффект. Подключается правополушарный канал восприятия, который, как известно, работает у большинства людей на порядок эффективнее, чем вербальный.

Метод *RAD* – один из наиболее известных способов быстро создавать прототипы[10].

RAD базируется на следующих базовых принципах:

- Эволюционное прототипирование;
- CASE-средства, как основной инструмент, включая возможности прямого и обратного проектирования и автоматической генерации кода;
- Высококвалифицированные специалисты, хорошо владеющие развитыми инструментальными средствами;
- Интерактивный JAD-метод, в котором общение совмещается с разработкой в режиме online;
- Жёсткие временные рамки, как противоядие от «расползания границ» проекта: если команда не укладывается в срок – функционал сужается.

Литература к лекции

1. Унифицированный процесс разработки программного обеспечения/ А. Якобсон, Г. Буч, Дж. Рамбо – СПб.: Питер , 2002. – 496 с.
2. Искусственный интеллект: в 3 книгах, кн. 2. Модели и методы / Справочник под ред. Э.В.Попова. - М.: Радио и связь. - 1990.
3. Марка Д.А. Методология структурного анализа и проектирования. – С.-Пб.: Питер, 1995. – 235 с.
4. Коберн А. Быстрая разработка программного обеспечения. – М.: Лори, 2002. 314 с.
5. Каменова, Громов. Моделирование бизнеса. Методология ARIS. — М.: Весть-МетаТехнология, 2001.
6. Марка Д.А. Методология структурного анализа и проектирования. – С.-Пб.: Питер, 1995. – 235 с.
7. Мацяшек Лешек, А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер. с англ. - М.: Издательский дом "Вильямс", 2002. - 432 с.: ил. - Парал. тит. Англ.
8. Орлик С., Булуй Ю. Введение в программную инженерию и управление жизненным циклом ПО Программная инженерия. Программные требования. Copyright © Сергей Орлик, 2004-2005. http://www.sorlik.ru/swebok/3-1-software_engineering_requirements.pdf
9. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. — 576с.: ил.

[10] Хотя задачи RAD на этом не ограничиваются

4. Формирование видения. Специфицирование требований

План лекции

Видение продукта и границы проекта
Концепция в ГОСТ РФ
Видение в RUP
Видение / рамки в MSF
Видение продукта и границы проекта
Акторы и варианты использования
Глоссарий
Спецификация варианта использования
Свободный формат
Шаблон полного описания варианта использования по А. Коберну
Табличные представления варианта использования
Шаблон варианта использования RUP
Выбор формы описания варианта использования
Спецификация нефункциональных требований
Атрибуты требований

Работы по формированию видения продукта и границ проекта обычно начинаются на самой ранней фазе проекта, до начала широкомасштабных консультаций по выявлению подробных требований, хотя в целом наличие и последовательность данных шагов зависит от выбранной методологии. На практике данные работы зачастую совмещаются. Правила извлечения требований, рассмотренные в лекции 3, могут быть использованы и при формировании видения.

Анализируя литературу по рассматриваемой тематике, можно выделить следующие широко употребимые ключевые слова: с одной стороны – концепция, видение, образ, с другой – рамки, границы, контекст.

В первом случае речь идёт о видении того, какой должна быть система. Обсуждаются высокоуровневые требования (возможности, свойства) продукта и наиболее существенные *ограничения*. Ряд авторов, напротив, настаивает на том, что видение должно быть «*ничем не ограниченным*».

Понятие видения широко употребимо в бизнес-анализе. Если у топ-менежмента компании имеется представление о том, какие ключевые цели, сегменты рынка, товарные позиции, прибыль должны быть достигнуты, допустим, через 5 лет – значит, компания имеет долгосрочное видение себя на рынке. Способ снятия ограничений при выработке видения позволяет выработать новый взгляд на вещи, «подняться над ситуацией», планировать будущее, отталкиваясь не от текущих ресурсов и ограничений, а от стратегических целей, применяя инновации, ноу-хау и т.п.

Данный опыт формирования видения во многом переносим и на процесс разработки информационных систем: нужно «увидеть» в горизонте средне- и (или) долгосрочного планирования, как АИС впишется в организационные процессы предприятия, какие ключевые выгоды она даст, какие проблемы позволит разрешить. При поиске новых методов и средств управления предприятием на основе информационных технологий зачастую приходится «перекраивать» существующие бизнес-процессы; по сути внедрение АИС, затрагивающей существенный процент процессов предприятия, неизбежно приводит к перестройке этих процессов с целью оптимизации деятельности предприятия, достижения ключевых факторов эффективности и пр.

Во втором случае (рамки, границы, контекст) обсуждаются такие вопросы, как граница системы и среды, требуемые ресурсы на создание системы, сроки. Построив «ничем не ограниченное видение», рано или поздно приходится вернуться к таким

прозаическим вещам, как бюджет, календарное планирование, подбор персонала, вехи проекта.

Всегда ли нужно создавать документ «Концепция»? Следует ли разделять видение и границы?

Зачастую Заказчик осознаёт необходимость автоматизации, как способ решения накопившихся проблем. Сформулировав для себя проблему, Заказчик часто видит и вариант её решения, с которым приходит к Исполнителю («мне нужен сайт», «требуется CRM-система» и т.п.). Квалифицированный Исполнитель не должен, сломя голову, спешить решать задачу в формулировке Заказчика. По образному выражению Г.Калянова[11] автоматизировать процессы «как есть» – всё равно, что асфальтировать дорожки, по которым ходят коровы.

В нотации RUP присутствует важная метафора: «Увидеть проблему за проблемой». Концепция как раз и служит для того, чтобы помочь Заказчику выявить именно те требования к системе, которые помогут ему оптимизировать работу своего предприятия в долгосрочной перспективе.

Поэтому этап формирования концепции важен, но он предъявляет и к Заказчику и к Исполнителю достаточно высокие требования: Заказчик должен выделить ресурсы и быть готовым к трудозатратам на совместный поиск решений; Исполнитель должен обладать достаточной квалификацией как в сфере IT-, так и в сфере управления предприятиями, чтобы разрабатываемое средство автоматизации действительно принесло пользу. Всё вышесказанное ничуть не исключает возможность работы без концепции: либо речь идёт о небольшом проекте, закладывать в бюджет которого этап выработки концепции просто нерентабельно, либо Заказчик сам обладает достаточной квалификацией, чтобы сформулировать требования к АИС, имея «концепцию в голове» и время для консультирования Разработчика.

Некоторые аргументы за разделение видения и границ были приведены выше. Провести чёткую границу между этими понятиями предлагает, в частности, процесс MSF. В конечном итоге, вопрос «разделять или не разделять» определяется выбранной методологией.

Рассмотрим основные требования к выработке концепции, заложенные в отечественных ГОСТ, методологиях RUP и MSF.

Концепция в ГОСТ РФ

В соответствии с ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания» [24], после этапа формирования (выявления) требований к системе выполняется этап разработки концепции системы.

Основные работы этапа:

- Изучение объекта;
- Проведение научно-исследовательских работ (НИР);
- Разработка вариантов концепции АС;
- Оформление отчёта о выполненной работе.

Так как данный этап хронологически стоит на втором месте, к его началу у Разработчика на руках уже имеется документ, в котором собраны основные требования пользователей.

Работы над концепцией начинаются с обследования объекта автоматизации. Выполняются НИР, направленные на исследование принципиальной реализуемости требований и возможных вариантов реализации.

---

1. [11] *Калянов Г. Н.* Консалтинг при автоматизации предприятий: Научно-практическое издание. Серия «Информатизация России на пороге XXI века». — М.: СИН-ТЕГ, 1997.

ГОСТ, в отличие от большинства современных методологий, в общем случае закладывает многоальтернативность вариантов концепции системы и планов их реализации. Каждый из проработанных вариантов оценивается с позиций требуемых ресурсов и функциональности. Для вариантов должны быть представлены оценки преимуществ и недостатков. Полезность проработки нескольких вариантов концепции заключается в том, что Заказчику трудно сформулировать самостоятельно видение системы, в то время, как выбор из набора вариантов, представленных Разработчиком – вполне посильная задача.

Кроме того, концепция должна отражать оценки качества, условия приёмки системы, оценку эффекта, ожидаемого от реализации. При оформлении отчёта необходимо привести обоснование предлагаемого варианта.

Видение в RUP

Шаги, которые необходимо пройти для формирования документа «Видение»:

- Формулировка проблем.
- Идентификация совладельцев
- Определение границ системы
- Идентификация ограничений
- Формулировка постановки задач
- Определение возможностей системы
- Оценка результатов

Для описания *проблем* предлагается шаблон, показанный в табл. 7-1.

Табл. 7-1.

Проблема	(описание проблемы)
Затрагивает	(совладельцы, затрагиваемые проблемой).
Ее следствием является	(каково влияние проблемы).
Успешное решение	(список некоторых ключевых преимуществ от успешного решения).

Идентификация *совладельцев* предполагает поиск и фиксацию интересантов проекта – представителей Заказчика и Исполнителя, инвесторов, внешних экспертов и пр.

Определение *границ системы* представляет собой нетривиальный процесс. Для этого используют контекстные диаграммы [23] (см. материалы 09-Моделирование требований). RUP в поиске границ предлагает отталкиваться от акторов и вариантов использования.

Среди источников *ограничений* обычно выделяют:

- Политические,
- Экономические,
- Среды,
- Технические,
- Выполнения,
- Системные.

Описание *возможностей* системы представляет собой формулировку высокоуровневых требований.

Шаблон документа «Vision» RUP содержит следующие основные разделы:

1. Введение
2. Позиционирование
3. Описания совладельцев и пользователей
4. Краткий обзор изделия
5. Возможности продукта
6. Ограничения
7. Показатели качества

8. Старшинство и приоритеты
9. Другие требования к изделию
10. Требования к документации
11. Приложение.

Во введении описываются цель документа, его контекст (связь и взаимовлияние с различными проектами), определения, акронимы и сокращения, ссылки на другие документы, краткое содержание.

В разделе «позиционирование» помещается определение решаемой проблемы (проблем), указывается целевой заказчик и исследуются деловые преимущества изделия перед аналогичными на рынке.

В описании совладельцев и пользователей, помимо собственно описания этих двух групп, исследуется демография рынка: целевые рыночные сегменты; размер и темпы роста рынка; существующие конкурентные предложения на рынке; репутация Разработчика на рынке;

Краткий обзор изделий содержит резюме изделия, описание его перспектив и ключевых возможностей, предположения и зависимости, указывается стоимость и её калькуляция, рассматриваются вопросы лицензирования и инсталляции.

В разделе, посвящённом возможностям продукта, они описываются более подробно, каждая – в отдельном параграфе.

В раздел «Ограничения» следует выносить существующие технические, технологические и др. обстоятельства, которые необходимо учитывать на данной стадии.

Раздел «Показатели качества» содержит описание наиболее существенных нефункциональных требований к системе (эффективности, надёжности, отказоустойчивости и др.).

Раздел «Старшинство и приоритеты» ранжирует сформулированные ранее требования и возможности системы по степени важности, очерёдности реализации и т.п.

Раздел «Другие требования к изделию» описывает применяемые стандарты, системные требования, эксплуатационные требования, требования к окружающей среде.

В требованиях к документации приводятся ключевые характеристики руководства пользователя, интерактивной справки, руководства по установке и конфигурированию, файла Read Me.

В приложение выносятся атрибуты возможностей. RUP рекомендует следующий набор атрибутов: статус, выгода, объём работ, риск, стабильность, целевой выпуск, назначение, причина.

Видение / рамки в MSF

Согласно белой книге MSF [25], на фазе выработки концепции (envisioning phase) закладывается одна из фундаментальных основ успеха проекта – **создание и сплочение проектной группы** на основе выработки единого видения. Проектная группа должна **четко представить себе**, что она хочет сделать для заказчика и сформулировать свою цель таким образом, чтобы максимально мотивировать как заказчика, так и саму проектную команду. Выработка высокоуровневого взгляда на цели и условия проекта может рассматриваться как ранняя форма планирования; она подготавливает почву для процессов создания детальных планов, которые будут осуществлены непосредственно во время фазы планирования.

Основными задачами фазы выработки концепции являются создание ядра проектной группы (см. ниже) и подготовка **документа общего описания и рамок проекта** (vision/scope document). Формирование видения проекта и специфицирование его рамок – не одно и тоже, хотя для успеха проекта необходимо и то, и другое. *Видение (vision)* – это ничем не ограничиваемое представление о том, каким должно быть

решение[12]. *Рамки (scope)* же дают четкие границы того, что из предложенного этим видением будет реализовано в условиях существующих проектных ограничений.

Управление рисками представляет собой итеративный процесс, осуществляемый на протяжении всего жизненного цикла проекта. Во время фазы выработки концепции проектная группа готовит документ оценки рисков и представляет главные риски проекта вместе с общим описанием и рамками проекта. Для получения дальнейшей информации об управлении рисками, см. “Белую книгу” дисциплины управления рисками MSF.

Также во время фазы выработки концепции производится выявление и анализ **бизнес-требований**. Более детально эти требования рассматриваются во время фазы планирования.

Ведущим ролевым кластером на фазе выработки концепции является “Управление продуктом”.

Шаблон MSF содержит следующие разделы:

- Бизнес-преимущества,
  - Описание преимуществ
  - Формулировка видения
  - Анализ выгод
- Концепция решения
  - Цели, задачи, предположения и ограничения
  - Анализ применимости
  - Требования
- Рамки
  - Список характеристик/функций
  - Вне рамок
  - Стратегия подготовки релизов
  - Критерии применимости
  - Эксплуатационные критерии
- Стратегии проектирования решения
  - Стратегия проектирования архитектуры
  - Стратегия технического проектирования

Акторы и варианты использования

Результатом выявления требований, см. материалы лекции 3 является реестр требований. Требования совладельцев обычно оформляются в простой письменной форме, без какой-либо особой регламентации. Типовой пример оформления требования к программе электронной почты – «Система должна позволять набирать текст сообщения с возможностью форматирования текста и вставки смайликов». Данные требования далеко не во всём могут удовлетворять критериям, сформулированным в лекции 2: они могут противоречить друг другу, быть неясными, неточными и т.д. Тем не менее, документ «Требования совладельцев», несмотря на невысокий уровень формализации, играет очень важную роль: в нём собраны мнения всех заинтересованных сторон и главная цель сбора начальных требований заключалась в том, чтобы получить по возможности как можно более полный набор требований, не пропустив чего-то важного.

Для того, чтобы повысить уровень информативности требований, устранить взаимные противоречия и добиться выполнения их других основных характеристик, осуществляется переход от полностью неформализованных текстов к частично регламентированным (например, шаблонами MS Word) текстам, классификация, присвоение наборов атрибутов, построение моделей, прототипирование.

Самым популярным и весьма эффективным способом повышения информативности требований является оформление их в виде вариантов использования (use case), предложенный И.Якобсоном (см., например, [26]).

Прежде, чем приступить собственно к специфицированию требований в форме вариантов использования, RUP рекомендует выявить реестр акторов[13] (actors) и вариантов использования.

Актор – это некто или нечто, обладающее активностью по отношению к программной системе. Если вы разрабатываете простой текстовый редактор, то, скорее всего, выбор актора не составит особого труда: это будет пользователь, набирающий текст. Однако не всегда всё так просто. Помимо пользователя в качестве актора может рассматриваться другая программная система, аппаратное устройство, в ряде случаев – активная компонента самой системы. Поиск акторов корпоративной информационной системы обычно сводится к анализу ролей различных пользователей. Менеджер по продажам, старший менеджер и начальник отдела продаж – один актор, два или три? Это зависит от их функциональных обязанностей, разграничения доступа, способов использования информационной системы. Поиск акторов может осуществляться, например методом мозгового штурма. В дальнейшем при необходимости найденные акторы могут обобщаться, пересматриваться и объединяться.

Вариант использования в первом приближении можно рассматривать, просто, как функцию, реализуемую системой. Однако, современный взгляд на организацию бизнеса говорит о том, что всякая функция должна иметь ценность для конечного потребителя продукта или услуги. Философия варианта использования предполагает выделение среди всего функционала системы подмножества, полезного конкретному конечному пользователю (точнее говоря, типу конечного пользователя). Другая сторона – вариант использования должен не только быть полезен, а ещё и позволять получать КП конкретные законченные результаты. Так, одной из функций текстового редактора, очевидно, является создание пустого файла. Но вряд ли КП будет использовать редактор с целью изготовления пустых файлов. Следовательно, создание пустого файла – функция, но не вариант использования системы. Вариантом использования может быть, например, подготовка в текстовом редакторе служебной записки. Вариант использования реализуется через функции системы.

Глоссарий

Помимо формирования требований совладельцев другим результатом начальной фазы выявления требований является концептуальный анализ проблемной области. Самым первым результатом его является формирование глоссария (словаря) основных используемых терминов. Значение глоссария трудно переоценить: он является основой, ключом для единообразного понимания описаний требований Заказчиком и Разработчиком.

Кроме того, глоссарий является отправной точкой для построения более развёрнутых моделей проблемной области, которые, на стадии реализации информационной системы, ложатся в основу объектной модели (для объектно-ориентированных приложений) и модели данных (для генерации схемы базы данных).

Глоссарий оформляется, как текст, состоящий из абзацев, каждый из которых определяет значение одного из терминов проблемной области. Термин обычно выделяют полужирным кеглем. Иногда проблемную область целесообразно сегментировать на ряд

---

[13] В различных русскоязычных текстах автору приходилось видеть следующие переводы термина «actor»: актор, актёр, актант, агент, субъект, пользователь. Так как все они либо неблагозвучны, либо неточны, за неимением лучшего далее по тексту будем пользоваться переводом, наиболее близким к транскрипции.

«подобластей» (subject areas). Тогда каждой из них в глоссарии выделяется отдельный параграф.

Спецификация варианта использования

Существуют различные шаблоны описания вариантов использования. Так, в монографии [27] рассматриваются следующие основные стили описания:

- Свободный формат,
- Полный формат (предложенный А. Коберном),
- Таблица в две колонки,
- Таблица в три колонки,
- Стиль RUP.

Кроме того, иногда целесообразно использовать:

- Псевдокод,
- Диаграмму активности UML (см. лекции 5),
- Другие графические модели.

Свободный формат

Свободный формат предполагает описание действий пользователя и системы в повествовательном стиле, например: «Менеджер запрашивает у Системы список заказов за период. Система отображает на экране найденные заказы данного Менеджера с указанием их основных атрибутов». Свободный стиль допускает использование конструкций «Если то». «Если Менеджер имеет полномочия Начальника Отдела, то Система предоставляет возможность просмотра заказов всех менеджеров этого отдела».

Шаблон полного описания варианта использования по А. Коберну

Название <краткая фраза в виде глагола в неопределённой форме совершенного вида, отражающая цель>

Контекст использования <уточнение цели, при необходимости – условия её нормального завершения>.

Область действия <ссылка на рамки проекта>. Например – подсистема бухгалтерского учёта.

Уровень <один из трёх: обобщённый, цели пользователя, подфункции>. Автор задаёт предопределённую трёхуровневую классификацию требований, в целом соответствующую классификации требований на бизнес-требования, требования пользователей и функциональные требования, см. материалы лекции 1.

Основное действующее лицо <имя роли основного актора или его описание>.

Участники и интересы <список других акторов-участников прецедента с указанием их интересов>.

Предусловие <то, что ожидается, уже имеет место>.

Минимальные гарантии <что гарантируется акторам-участникам>. Например – в случае неудавшейся транзакции все данные, имевшиеся в системе до её начала, сохраняются неизменными.

Гарантии успеха <что получат акторы-участники в случае успешного достижения цели>.

Триггер <то, что «запускает» вариант использования, обычно – событие во времени>.

Основной сценарий <здесь перечисляются шаги основного сценария, начиная от триггера и вплоть до достижения гарантии успеха>.

Формат описания: <Номер шага> <Описание действия>

Расширения <здесь последовательно описываются все альтернативные сценарии>. Каждая из альтернатив привязана к шагу основного сценария.

Формат описания: <Номер шага.Номер расширения> <Условие>:<Действие или ссылка на подчинённый вариант использования>.

Любой из шагов основного сценария может иметь 1 или более ветвлений. Каждое ветвление оформляется в виде расширения. В блоке «Расширения» все расширения описываются последовательно.

В случае, если альтернативный сценарий не удаётся описать одной строкой – применяется следующий формат.

Начиная со строки, следующей после описания расширения, идёт описание его действий в формате основного сценария:

<Номер шага.Номер расширения.Номер шага расширения> <Действие>

Описание расширения заканчивается описанием выхода из расширения. Основные варианты выхода из расширения: возврат к очередному по номеру шагу основного сценария, окончание прецедента, переход к другому шагу основного сценария.

Список изменений в технологии и данных <что гарантируется акторам-участникам>. Например – в случае неудавшейся транзакции все данные, имевшиеся в системе до её начала, сохраняются неизменными.

Вспомогательная информация <дополнительная информация, полезная при описании варианта использования>.

Табличные представления варианта использования

Иногда представляется удобным помещать сценарии вариантов использования в таблицу, как это показано ниже. Информация при этом принимает более структурированный вид.

Таблица в 2 колонки:

Актор	*Действие*
Пользователь	Формирует запрос на поиск заказов
Система	Отображает список заказов
Пользователь	Выбирает требуемый заказ
Система	Показывает подробную информацию по заказу

Таблица в 3 колонки:

№ шага	*Пользователь*	*Система*
1	Делает запрос на поиск заказов	Отображает список заказов
2	Выбирает требуемый заказ	Показывает подробную информацию по заказу

Шаблон варианта использования RUP

С шаблоном описания варианта использования RUP и примерами можно ознакомиться в интерактивной версии RUP[14].

Ниже приведён краткий обзор его разделов.

1. **Наименования и краткое описание.** В этом разделе указывается: наименование варианта использования, акторы варианта использования, краткое (в один абзац) описание варианта использования.

2. **Поток событий**

2.1. **Основной поток событий**

Так же, как в «Основной сценарий».

2.2. **Альтернативные потоки событий**

[14] http://www-306.ibm.com/software/rational/

Каждый из альтернативных сценариев описывается в отдельном параграфе, в том же стиле, что и основной поток событий. Альтернативные сценарии описывают поведение системы при любых отклонениях от основного сценария, а также поведение в исключительных ситуациях.

3. **Специальные требования**

Здесь перечисляются нефункциональные требования, имеющие непосредственное отношение именно к этому варианту использования.

4. **Предусловия**

События, описываемые предусловиями или постусловиями, должны быть состояниями, которые пользователь может наблюдать [15**Ошибка! Источник ссылки не найден.**]. Предусловие описывает состояние, в котором система должна находиться до начала исполнения прецедента.

5. **Постусловия**

Постусловие RUP по сути описывает то же, что и минимальная гарантия у Коберна. Л.Новиков [15**Ошибка! Источник ссылки не найден.**] акцентирует внимание на том, что корректно сформулированное постусловие должно быть истинным при любом возможном сценарии прецедента, а не описанном в основном потоке.

6. **Точки расширения**

Данный параграф определяет положение точек, расширяющих поток событий.

Выбор формы описания варианта использования

При выборе формы и степени подробности варианта использования следует учитывать такие факторы, как:

- Размеры проекта,
- Важность проекта и варианта использования,
- Традиции, сложившиеся в коллективе «Заказчик-Разработчик».

Для небольшого проекта вряд ли будет целесообразным применять описания вариантов использования в развёрнутом формате, достаточно использовать краткую форму свободного стиля. Для проекта, в котором задействовано более десяти участников, в котором возникают проблемы разбиения на микро-коллективы, координации участников, следует выбрать более формализованный и более подробный вариант.

Степень подробности зависит также от критичности проекта в целом и критичности варианта использования в данном проекте. А.Коберн делит все программные проекты по степени критичности на 4 категории: исходя из цены ошибок: «проекты, ошибки в которых могут привести к…»:

- опасности для жизни людей,
- невосполнимым финансовым потерям,
- финансовым потерям в ограниченном объёме,
- снижению комфортности конечного пользователя.

Очевидно, что военные системы, либо системы управления сложными техническими объектами требуют более скрупулёзного документирования, в том числе – и на уровне описания вариантов использования.

Кроме того, в одном и том же проекте могут встречаться более важные – с позиций частоты и массовости использования, сложности для понимания, технических рисков и т.д. и менее важные прецеденты. В этом случае для разных прецедентов одного и того же проекта вполне допустимо описание с разной степенью подробности.

Наконец, спецификация вариантов в стиле Коберна, стиле RUP, в табличной форме, с использованием псевдокодов или графических конструкций (см. материалы 09-

Моделирование требований) во многом определяется субъективным выбором автора прецедентов и сложившимся опытом работы с заказчиком проекта.

Спецификация нефункциональных требований

Описание нефункциональных требований обычно осуществляется в форме, близкой к свободному формату описания варианта использования. RUP рекомендует концентрировать нефункциональные требования в документе, описывающем вариант использования во всех случаях, когда это возможно. В случае, если нефункциональные требования носят общий характер и не могут быть привязаны к конкретному прецеденту – они выносятся в документ «Дополнительная спецификация».

Атрибуты требований

Описания требований должны быть операбельны. Для этого все требования должны учитываться в той или иной учётной системе, будь то электронная таблица MS Excel, специализированная база данных, либо интегрированная среда управления изменениями. При регистрации требования оно проходит классификацию в соответствии с определённой системой признаков. Основные признаки (атрибуты) требований были рассмотрены в лекции 1. Кроме того, для оперативного управления требованиями бывает полезно назначить им такие свойства, как проект, ответственное лицо, статус, риск, степень законченности и т.п. В RUP для управления атрибутами требований предусмотрен артефакт «Атрибуты требований».

Артефакт «Атрибуты требований», предлагаемый RUP, представляет собой репозиторий текстов требований, их атрибутов и трассируемости.

Атрибуты требований представлены матрицей атрибутов требований, где для каждого типа требований перечисляются требования по одной оси и атрибуты требований этого типа по другой. Для каждого требования указываются значения его соответствующих атрибутов. Примеры атрибутов: статус во времени, приоритет, важность, риск, № итерации (этапа) в плане.

Трассируемость описывается в виде дерева, показывающего в графическом виде входящие и (или) исходящие связи трассируемости (см. материалы лекции 7).

Литература к лекции

23. Марка Д., МакГоуэн К. Методология структурного анализа и проектирования. — М.: МетаТехнология, 1993.
24. ГОСТ 34.601-90. Информационная технология. Автоматизированные системы. Стадии создания.
25. Белые страницы MSF. http://www.microsoft.com/rus/msdn/msf
26. Фаулер М.: издательство «Лори», 2002. – 263., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.:Мир, 1999. – 191 с., ил.
27. Алистер Коберн. Современные методы описания функциональных требований к системам.
28. Л.Новиков. Введение в Rational Unified Process. http://www.interface.ru/rational/interface/151199/rup/main.htm

5. Расширенный анализ требований. Моделирование и прототипирование

План лекции

Какие модели использовать
Модели UML, поясняющие функциональность системы
Диаграмма вариантов использования
Диаграмма действий, диаграмма состояний
Диаграммы UML, поясняющие внутреннее устройство системы
Альтернативные языки моделирования
Диаграмма потоков данных
Другие виды моделей
Цели прототипирования
Классификация прототипов
Горизонтальный и вертикальный прототипы
Одноразовый и эволюционные прототипы
Бумажный прототип. Раскадровка
Иллюстрированные сценарии прецедентов
Ориентиры
Средние значения атрибутов и объёмы объектов
Средняя интенсивность использования

Какие модели использовать

Вербальные описания вариантов использования системы, рассмотренные в предыдущей лекции, на сегодня являются стандартом де-факто для формулировки соглашения между Заказчиком и Исполнителем. Если обе стороны готовы выделить достаточное количество времени на внимательный и всесторонний анализ требований к системе и на начальной фазе её создания сформулировали 80% всех возможных сценариев использования системы на понятном сторонами языке – значит, ключевые риски создания системы – риск различного понимания проблемы и риск размытия границ во многом преодолены.

Однако, далеко не всякий Заказчик готов скрупулёзно обсуждать скучные тома описания вариантов использования, которые даже для систем среднего размера могут достигать сотни страниц.

Чтобы облегчить процесс формулировки и понимания требований для Заказчика, существует ряд приёмов. Во-первых, требования можно формулировать на разных уровнях абстракции. Так, уровень описания требований, поддерживаемый в документе «Видение», является достаточно сбалансированным. То же можно сказать и про краткие (в один абзац) описания ключевой функциональности системы. Действуя таким образом, мы, очевидно, решим проблему вовлечения Заказчика в анализ задач, однако указанные выше риски будут снижены недостаточно: концептуальные описания функциональности оставляют много свободы для толкования в ту или иную сторону.

Хорошим подспорьем в решении задачи является применение визуальных средств описания требований. Как известно, у большинства людей правополушарное (образное) мышление позволяет воспринимать информацию в разы и порядки более ускоренном темпе, чем левополушарное (вербальное).

На сегодня в арсенале аналитика существуют десятки методик, языков, визуальных представлений, позволяющих моделировать требования к системе. При создании информационных систем стандартом де-факто является универсальный язык моделирования, UML [14,15]. Некоторые другие нотации были упомянуты в лекции 3.

Как уже отмечалось в лекции, посвящённой анализу контекста АТ, процесс анализа требований тесно связан, с одной стороны, с анализом проблемной области, с другой – с архитектурным анализом и проектированием. Часто на практике бывает трудно вычленить границы компетенций этих потоков работ. Так, модель анализ потоков данных, широко используемая в анализе проблемной области, упоминается многими авторами, как модель, полезная в анализе требований. Ряд исследователей считает целесообразным иллюстрировать описания требований диаграммами классов, хотя, строго говоря, выделение классов относится не к анализу требований, а к архитектурному анализу.

Как определить целесообразность использования тех или иных приёмов, методик, языков моделирования при анализе требований? Здесь можно предложить три практические рекомендации.

Во-первых, анализ требований призван изучать взаимодействия автоматизированной информационной системы и её среды, т.е. пользователей, сетевых и системных компонент, находящихся вне системы. Следовательно, бизнес-модели, описывающие взаимодействия между компонентами организационной системы, строго говоря, можно рассматривать лишь как «сырьё» для извлечения требований, но не как модели, описывающие требования.

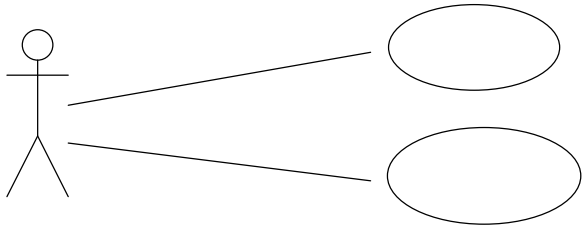
Во-вторых, анализ требований должен находить ответ на то, ЧТО делает система, абстрагируясь от деталей реализации, т.е. того, КАК она это делает. Поэтому, допустим, диаграмма взаимодействия объектов, реализующих тот или иной вариант использования, можно рассматривать скорее, как иллюстрацию внутреннего устройства системы, полезную для программиста, чем модель для заказчика.

Однако, наиболее важным является третье соображение, в чём-то «оппозиционное» по отношению к первым двум. Для моделирования анализа требований следует применять модели, наиболее адекватно проясняющие функциональность системы и особенности её использования. Однако, аналитик волен выбирать те языки и методики, которые позволят добиться целевой функции: снижения рисков непонимания между Исполнителем и Заказчиком и размытия границ. Поэтому, иллюстрируя варианты использования, начинайте с «канонических» способов, которые будут рассмотрены чуть ниже, но, если посчитаете целесообразным отклониться от них – экспериментируйте смело.

Модели UML, поясняющие функциональность системы

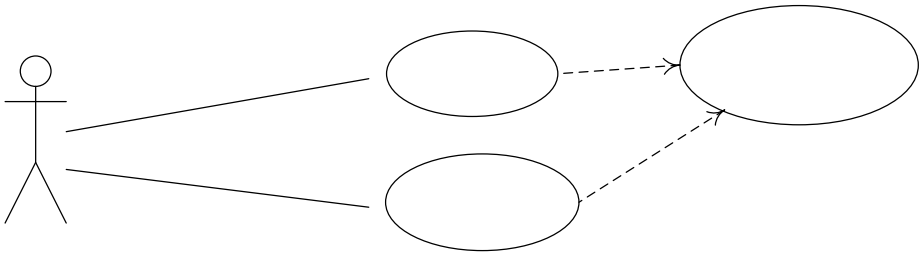
Диаграмма вариантов использования

Диаграмма вариантов использования UML, Use Case Diagram – одно из самых простых представлений системы. Её базовые «строительные элементы» – акторы и варианты использования. Диаграмма задумана так, чтобы дать наиболее общее представление о функциональности системы (её компоненты), не вдаваясь в детали взаимосвязей функций. Поэтому основной вид отношения, используемый в диаграмме – ассоциация между актором и вариантом использования.

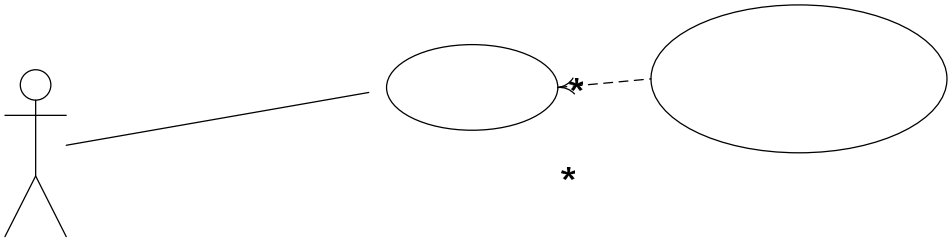


Другие виды отношений – отношение включения (include), расширения (extend) и обобщения/генерализации.

Включение служит для обозначения подчинённых вариантов использования (когда один или более вариантов использования содержат вызовы одной и той же функциональности).



Расширение в точности соответствует точке расширения, используемой при описании варианта использования, см. материалы лекции 4.



Менеджер

Отношение обобщения может применяться как к акторам, так и к вариантам использования, с целью указания специализации одних относительно других.

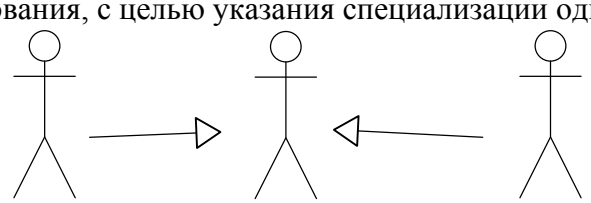


Диаграмма действий

Если диаграмма вариантов использования даёт «вид сверху» на функциональность системы, диаграмма действий UML, напротив, позволяет подробно иллюстрировать отдельный вариант использования и его сценарии.

Основные компоненты описания системы:

- Функции (действия),
- Символы «старт» и «стоп»,
- Потоки управления,
- Разветвители,
- Линейки синхронизации.

Менеджер

Менеджер

Пользователь

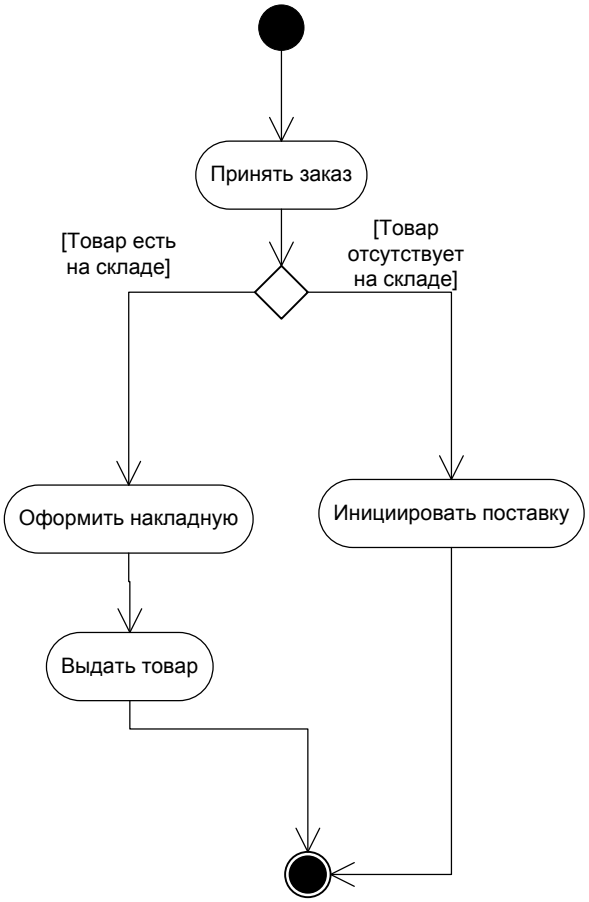


Диаграмма действий позволяет проиллюстрировать вариант использования с требуемой степенью подробности. Линейный вариант использования приводит к диаграмме действий с линейным потоком управления между действиями. Действия варианта использования с альтернативными сценариями реализуется через разветвители. Линейки синхронизации позволяют описывать такие сложные конструкции, как синхронизацию начала (окончания) параллельных во времени процессов.

Помимо стандартного формата описания, UML предлагает вариант с «плавательными дорожками». Этот формат удобен для описания случая, когда в варианте использования участвуют несколько акторов.

Диаграмма состояний

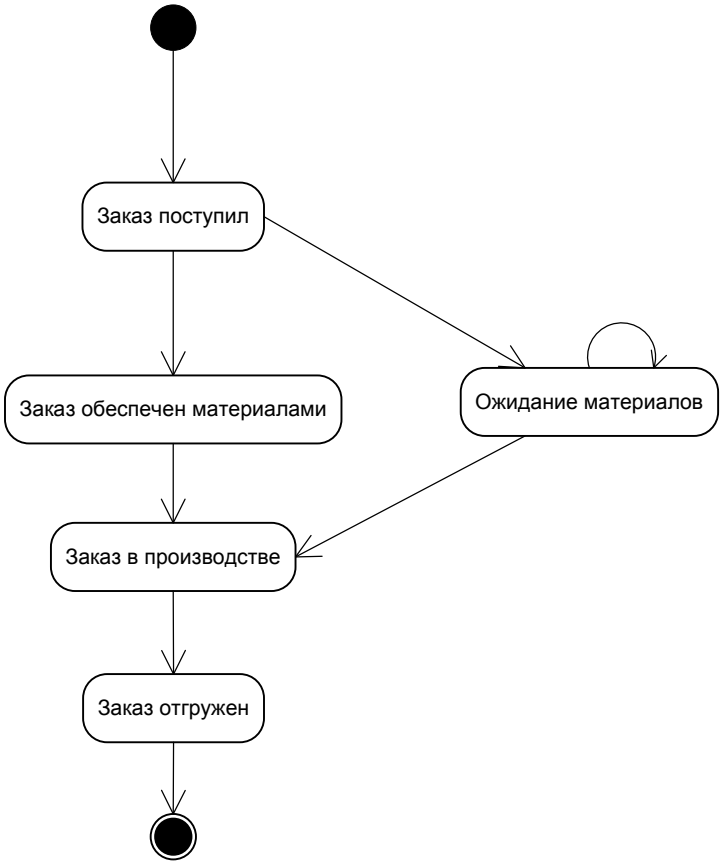
Диаграмма состояний в анализе требований используется, когда требуется исследовать поведение системы, как конечного автомата. Это представление пришло в UML из теории систем.

В общем случае диаграмма состояний описывает, как система себя ведёт в более, чем одном варианте использования. Синтаксис диаграмм состояний во многом совпадает с синтаксисом диаграмм действий.

Основные компоненты описания системы:

- Простые состояния,
- Составные состояния,
- Символы «старт» и «стоп»,
- Переходы,
- Линейки синхронизации.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия [15]. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.



Переход системы из состояния в состояние осуществляется при наступлении *событий*. При этом говорится, что переход *срабатывает*. Переход может быть безальтернативным, либо содержать альтернативы. Во втором случае переход обусловлен наступлением *сторожевых условий*. Наконец, событие может сопровождаться выражением действия, которое происходит в случае, если срабатывает переход. Полный синтаксис описания перехода (надписи на стрелке) следующий:

Событие [сторожевое условие] / выражение действия

Иногда бывает полезным объединить часть состояний в одно мета-состояние. Графически это выглядит, как символ состояния (прямоугольник со скруглёнными углами), содержащий внутри себя несколько символов состояний. При этом возможны переходы между подчинёнными состояниями, переходы между подчинённым и внешним состояниями и переходы между составным и внешним состоянием.

Диаграммы UML, поясняющие внутреннее устройство системы

Некоторые авторы [16] рекомендуют использовать при описании требований диаграммы UML, описывающие создаваемую систему через её компоненты (классы, объекты), отношения и взаимодействия между ними. Данный подход имеет свои ограничения (см. Принцип 2).

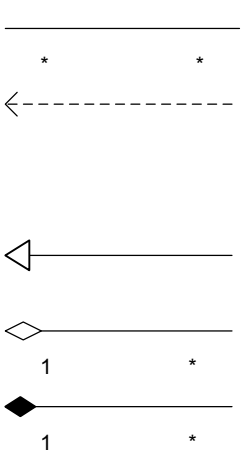
Диаграмма классов. Для создания диаграммы классов необходимо:

1. Осуществить поиск классов (ключевых компонент проблемной области).
2. Для каждого найденного класса определить его имя, основные атрибуты, операции и (или) ответственности.
3. Исследовать отношения найденных классов.

Классы на диаграмме представляются в виде прямоугольников, отношения – в виде линий, связывающих прямоугольники. Линии разного типа графически отличаются различной штриховкой и стрелками.

Принято выделять [14] 3 уровня абстракции классов: концептуальный уровень, уровень спецификации, уровень реализации. Анализ требований разумно сопровождать диаграммами, отражающими концептуальный уровень. На данном уровне при описании классов целесообразно указать их наименования и ответственности. Атрибуты и операции можно не указывать, либо ввести только самые основные, отложив их исследование на более поздние стадии детализации.

Отношения, подлежащие анализу на концептуальном уровне – это:



ассоциация (именованная связь),
зависимость (изменения в одном классе приводят к изменениям в другом),
обобщение / генерализация (родовидовое отношение),
агрегация (отношение «часть-целое»),
композиция (отношение «часть-целое», однозначно регламентирующее количество и состав частей целого).

Диаграмма классов показывает статическую структуру проблемной области. Для анализа взаимодействия объектов – экземпляров класса в ходе реализации варианта использования в UML предусмотрены две диаграммы взаимодействия: диаграмма кооперации и диаграмма последовательности.

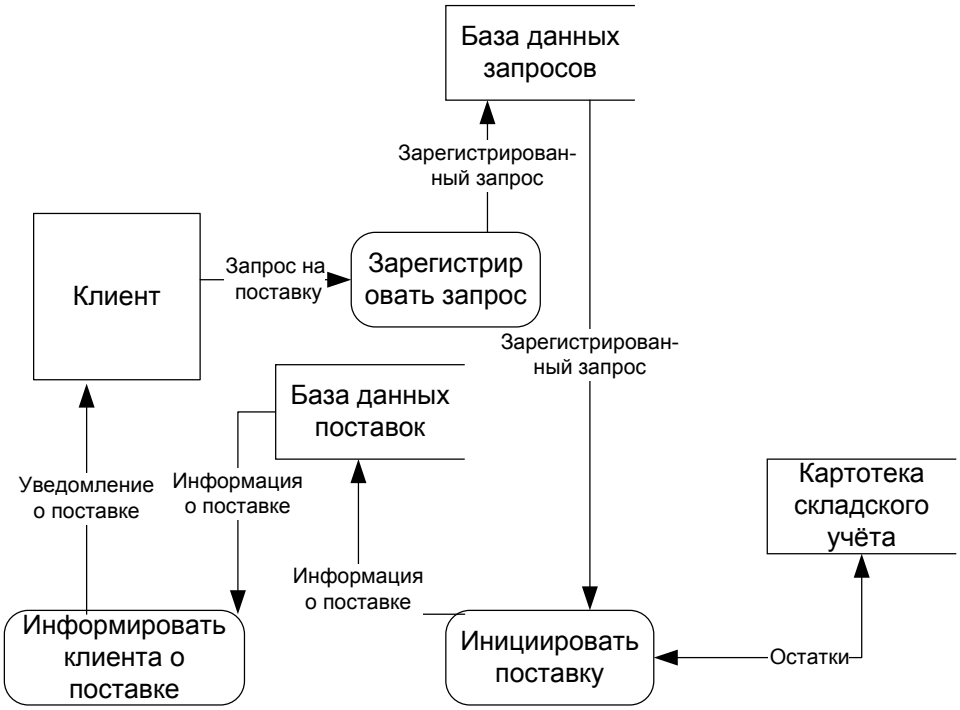
По мнению автора, если диаграмма классов в ряде случаев и может рассматриваться, как артефакт, поясняющий структуру проблемной области, то диаграммы взаимодействия вряд ли стоит рассматривать в качестве выразительного языкового средства, иллюстрирующего требования к системе в диалоге «Заказчик-Исполнитель». Тем не менее, в соответствие с Принципом 3 свободы выбора языковых средств, аналитик, решивший использовать данные диаграммы, может ознакомиться с ними в специальной литературе [14-15].

Альтернативные языки моделирования

Диаграмма потоков данных

Диаграмма потоков данных (data flow diagram, DFD) – один из основных инструментов структурного анализа и проектирования информационных систем, существовавших в «доюмээльную» эпоху. Несмотря на имеющее место в современных условиях смещение акцентов от структурного к объектно-ориентированному подходу к анализу и проектированию систем, «старинные» структурные нотации по-прежнему широко и эффективно используются как в бизнес-анализе, так и в анализе информационных систем.

Исторически сложилось так, что для описания диаграмм DFD используются две нотации – Йодана (Yourdon) и Гейна-Сарсона (Gane-Sarson), отличающиеся синтаксисом. На приведённой ниже иллюстрации использована нотация Гейна-Сарсона.



Информационная система принимает извне *потоки данных*. Для обозначения элементов среды функционирования системы используется понятие *внешней сущности*. Внутри системы существуют *процессы* преобразования информации, порождающие новые потоки данных. Потоки данных могут поступать на вход к другим процессам, помещаться (и извлекаться) в *накопители данных*, передаваться к внешним сущностям.

Модель DFD, как и большинство других структурных моделей – иереархическая модель. Каждый процесс может быть подвергнут декомпозиции, т.е. разбиению на структурные составляющие, отношения между которыми в той же нотации могут быть показаны на отдельной диаграмме. Когда достигнута требуемая глубина декомпозиции – процесс нижнего уровня сопровождается *мини-спецификацией* (текстовым описанием).

Кроме того, нотация DFD поддерживает понятие *подсистемы* – структурной компоненты разрабатываемой системы.

Нотация DFD – удобное средство для формирования *контекстной диаграммы,* т.е. диаграммы, показывающей разрабатываемую АИС в коммуникации с внешней средой. Это - диаграмма верхнего уровня в иерархии диаграмм DFD. Её назначение – ограничить рамки системы, определить, где заканчивается разрабатываемая система и начинается среда. Другие нотации, часто используемые при формировании контекстной диаграммы – диаграмма SADT[15], диаграмма Диаграмма вариантов использования.

Другие виды моделей

Среди многообразия моделей, использующихся в анализе систем, хочется особо отметить ещё две нотации, позволяющие описать сложные многоальтернативные взаимодействия компонент информационной системы – нотацию IDEF3 [17] и eePC-диаграмму ARIS [6].

Для моделирования требований к системам с разветвлённой логикой К.Вигерс рекомендует использовать таблицы и деревья решений [8]. Часто на практике бывают полезны диаграммы сущность-связь и SADT-диаграммы [17].

[15] Марка Д.А. Методология структурного анализа и проектирования. – С.-Пб.: Питер, 1995. – 235 с.

Цели прототипирования

Всё то, что говорилось в предыдущей лекции об особенностях восприятия человеком вербальной и невербальной информации по отношению к моделям, в ещё большей степени следует отнести и к визуальным прототипам.

Рассмотрим основные цели, требующие применения прототипов [8-21]:

- прояснить неясные требования к системе;
- выбрать одно из различных концептуальных решений;
- проанализировать осуществимость.

1. **Неясные требования.** Часто Заказчику бывает трудно сформулировать требования к тому, что он ожидает от системы. В этом случае прототип интерфейса пользователя (User Interface, UI), оперативно созданный по результатам интервью, даёт ему возможность увидеть схематичную реализацию того, как Исполнитель увидел соответствующую часть системы. Что интересно – в данном случае полезен любой исход прототипирования: если Исполнитель понял требования хорошо – польза очевидна; если не очень – польза заключается в том, что Заказчик может указать, в чём заключается непонимание, тем самым решив основную задачу – сделать неясное ясным.

2. **Разные варианты решения.** Любую техническую задачу можно решить различными способами. Это касается как задачи формулировки требований, так и её реализации в UI.

Рассмотрим пример. Снабженцу поступает входной поток требований на комплектацию заказов материалами. Разные позиции одного и того же требования могут быть закуплены у различных поставщиков. Снабженец должен сопоставить поставщика каждой позиции каждого из требований. Есть как минимум два сценария решения этой задачи.

А) Сценарий последовательной обработки требований.

А1. Система отображает реестр требований, имеющихся во входной очереди.

А2. Пользователь выбирает очередное требование.

А3. Система отображает перечень материалов требования и справочник поставщиков.

А4. Пользователь сопоставляет каждой из позиций требования поставщика из справочника поставщиков.

А5. Система придаёт требованию статус «обработано», высылает по электронной почте автору требования уведомление.

А6. Продолжать с шага А1, пока очередь не опустеет.

Б) Сценарий группировки по материалам.

Б1. Система отображает позиции всех требований и справочник поставщиков.

Б2. Пользователь группирует позиции по типу (так, чтобы однотипные позиции, поставляемые одним и тем же поставщиком, находились рядом).

Б3. Пользователь выбирает группу позиций и сопоставляет ей поставщика.

Б4. Система проверяет – не появились ли полностью обработанные требования. При положительном исходе проверки присваивает этим требованиям статус «обработано» и высылает по электронной почте автору требования уведомление.

Б5. Продолжать с шага Б1, пока очередь не опустеет.

Первый сценарий удобен тем, что позволяет снабженцу работать в разрезе авторов требований, начать с самых критичных по времени требований, контролировать процесс их обработки. Второй сценарий удобен тем, что позволяет одновременно наблюдать на экране строки разных требований, объединяя их в единый заказ.

После реализации прототипов UI по первому и второму сценариям Заказчик, оценив их достоинства и недостатки, смог в диалоге с Разработчиком сформулировать третий, комбинированный сценарий, сочетающий достоинства первых двух.

3. **Анализ осуществимости.** Часто бывает так, что комбинация функциональных, нефункциональных требований и ограничений такова, что возникает риск невозможности их реализации. Как правило, такой риск связан с требованиями к быстродействию системы при известных ограничениях среды её реализации. В этом случае создаются прототипы (не обязательно, связанные с UI), реализующие соответствующую часть системы, имитирующие потоки данных, поступающие на её вход и их обработку.

Классификация прототипов

Вслед за К. Вигерсом [8] рассмотрим следующие классификации прототипов:

- горизонтальные и вертикальные;
- одноразовые и эволюционные;
- бумажные и электронные, раскадровки[16].

Горизонтальный прототип

Горизонтальный или поведенческий прототип (horizontal prototype, behavioral prototype) моделирует интерфейс пользователя приложения, не затрагивая логику обработки и базу данных.

Если в разработанном интерфейсе используются фрагменты базы данных – они имитируются в программном коде. При этом тексты, отображаемые на экране, должны отражать реальную специфику проблемной области, иначе пользователю будет трудно сосредоточиться. Если при нажатии на элемент управления должны производиться какие-то расчёты или запросы во внешние системы – результаты запросов также имитируются. Желательно реализовать ту часть кода, которая отвечает за перемещение между экранами в процессе исполнения вариантов использования, чтобы пользователь смог понять, как будет действовать система в ответ на его действия.

Горизонтальные прототипы следует использовать для достижения цели прояснения неясных, либо многоальтернативных требований.

Вертикальный прототип

Вертикальный или структурный прототип (vertical prototype, structural prototype) не ограничивается интерфейсом пользователя. Он реализует вертикальный «срез» системы, затрагивая все уровни её реализации. При создании такого рода прототипов рекомендуется использовать те языки и среды реализации, что и при изготовлении целевой системы (что, вообще говоря, совсем не обязательно для горизонтальных прототипов).

Основные цели применения такого рода прототипов – анализ применимости, проверка архитектурных концепций.

Одноразовый прототип

Одноразовый или исследовательский прототип (throwaway prototype, exploratory prototype) создаётся, когда нужно быстро промакетировать те или иные аспекты и компоненты системы.

Целям создания исследовательских прототипов служит технология RAD (rapid application development) – быстрая разработка приложений[17], см. материалы лекции 3.

---

[16] Понятие раскадровки [22] отсутствует в классификации Вигерса, но хорошо её дополняет.

[17] Строго говоря, данная технология разрабатывалась не только для целей создания одноразовых прототипов, но и для реально действующих систем. Однако, приложения, созданные по технологии RAD на современном уровне её развития, редко бывают жизнеспособными, в первую очередь за счёт недостаточного быстродействия.

Одноразовый прототип должен создаваться быстро. При его разработке не следует уделять внимание вопросам повторного использования кода, качества, быстродействия, технологичности и т.п.

В результате получается «сырой» код, который может содержать значительное количество дефектов. Необходимо принять меры к тому, чтобы фрагменты кода, реализующие такого рода прототипы, не стали частью целевой системы.

К. Вигерс приводит следующую схему перехода от одноразового прототипа к детально проработанному UI:

На рисунке присутствует новое, не раскрытое ранее не понятие: «карта диалога», говорят также «схема диалога». Прежде, чем создавать горизонтальный прототип, необходимо определиться – какие основные экраны будут присутствовать, какие окна будут открываться, какие правила перехода между ними будут поддерживаться. Информация такого рода хорошо ложится на модель диаграммы состояний, см. материалы лекции 5, где разным экранам (окнам) сопоставляются состояния, а активным элементам управления, вызывающим закрытие одних интерфейсных элементов и открытие других – переходы.

Эволюционный прототип

Эволюционный прототип (evolutionary prototype) создаётся, как первое приближение системы, призванное стать впоследствии самой системой.

Код эволюционного прототипа должен последовательно, в течении одной или более итераций, перерасти в код целевого приложения. Поэтому данный вид прототипов требует всего того, от чего следует отказаться при создании одноразовых прототипов: скрупулёзной разработки, применения технологических методов и приёмов, тестирования результатов и т.п.

В таблице приведено соотношение между рассмотренными выше 4 видами прототипов [8].

	Одноразовые	*Эволюционные*
Горизонтальные	▪ Прояснение и уточнение примеров использования и функциональных требований ▪ Выявление пропущенных требований ▪ Исследование возможных вариантов интерфейса пользователя	▪ Реализация базовых вариантов использования ▪ Реализация дополнительных вариантов использования по приоритетам ▪ Реализация и доработка web-сайтов ▪ Адаптация системы к быстро меняющимся требованиям бизнеса
Вертикальные	▪ Демонстрация технической осуществимости	▪ Реализация и наращивание ключевой клиент-серверной функциональности и уровней коммуникации ▪ Реализация и оптимизация основных алгоритмов ▪ Тестирование и настройка

		производительности

Бумажный прототип

Бумажный прототип (paper prototype) – отличная альтернатива рассмотренным выше разновидностям электронных прототипов в случае, когда Разработчик ограничен в ресурсах. Наброски интерфейсов на бумаге, конечно, не заменят интерфейс, созданный в среде разработки. Однако, при всех недостатках, у таких прототипов есть два существенных достоинства.

1. Заказчик не станет акцентировать внимание на цветовом решении, форме кнопок и т.п., отвлекаясь от анализа функциональности.

2. Заказчик никогда не скажет, глядя на бумажный интерфейс: «Да вы, я вижу, уже создали систему на 85%! Давайте закончим её в течении недели».

Раскадровка

Решением промежуточного между электронным и бумажным вариантами прототипов UI класса, является презентации, изготовленные при помощи средств электронного офиса (например, комбинации Microsoft Visio и Microsoft PowerPoint). В этом случае пользователь лишён свободы выбора, предоставляемой ему поведенческим прототипом. Но идею пошаговой смены экранов в процессе реализации сценария варианта использования вполне можно реализовать. Данный вид решения определяется в [22], как *пассивная раскадровка*. *Активная раскадровка* является дальнейшим развитием понятия пассивной раскадровки, с применением средств анимации и т.п. Третий вид раскадровки, вводимый в [22] – *интерактивная* представляет собой электронный одноразовый горизонтальный прототип.

Иллюстрированные сценарии прецедентов

Иллюстрированные сценарии прецедентов, ИСП [15], наряду с прототипами позволяют достичь лучшего понимания между Заказчиком и Разработчиком. Но если прототипы адресованы скорее Заказчику, нежели Разработчику, то с ИСП ситуация обстоит наоборот: они содержат дополнительные сведения, помогающие Разработчику лучше понять специфику проблемной области и, тем самым, лучше отразить её в интерфейсе пользователя.

Основная идея ИСП – «разбавить» текст описания сценария варианта использования *аспектами применимости.*

Аспект применимости – информация, позволяющая расширить описание прецедента описаниями, конкретизирующими те или иные его особенности и, в конечном итоге, повысить степень комфортности пользователя.

Различают [15] 3 разновидности аспектов применимости: ориентиры, средние значения атрибутов и объёмы объектов, средняя интенсивность использования.

Ориентиры

Ориентиры – это описание опциональных функциональных возможностей системы. Отсутствие таких возможностей не приводит к фатальной неудаче. Присутствие – улучшает применимость, снабжая полезной информацией. Ориентиры следует расценивать не как требования, а как пожелания или рекомендации.

Пример. Описание потока событий ИСП для прецедента «Оформить заказ», расширенного ориентирами (текст в квадратных скобках).

В процессе выполнения прецедента менеджер по приёму заказов выбирает заказчика из клиентской базы, определяет товарные позиции из справочника и указывает их количество. Система отображает на мониторе наименование позиций, цену, сумму и

количество на складе. Менеджер назначает скидку и определяет порядок оплаты. Система рассчитывает итоговую сумму. [Менеджер должен иметь возможность видеть текущее сальдо расчётов с клиентом и данные по последним десяти сделкам со статистикой по дисциплине соблюдения договорных обязательств].

Средние значения атрибутов и объёмы объектов

Данная информация позволяет оптимальнее построить пользовательский интерфейс и оценить на ранних стадиях проекта «узкие места» в обработке данных, которые могут повлиять на производительность системы.

Так, при выборе из 2 возможностей лучше подойдёт элемент управления checkbox, при выборе, ограниченном 2-3 десятками позиций – выпадающий список, при многообразии, измеряемом тысячами вариантов, потребуются дополнительные средства фильтрации и поиска.

Пример. Описание потока событий ИСП для прецедента «Оформить заказ», расширенного объёмами и средними значениями объектов (текст в фигурных скобках).

В процессе выполнения прецедента менеджер по приёму заказов выбирает заказчика из клиентской базы {до 10000 клиентов}, определяет товарные позиции из справочника {товары разбиты на 10 категорий, количество позиций в категории не превышает 500} и указывает их количество {до 100 позиций, средняя закупка – 8 позиций}. Система отображает на мониторе наименование позиций, цену, сумму и количество на складе. Менеджер назначает скидку и определяет порядок оплаты {на данный момент существуют 3 варианта порядка оплаты}. Система рассчитывает итоговую сумму.

Средняя интенсивность использования

Средняя интенсивность использования позволяет выделить сценарии «массового» использования, в которых всё должно быть идеально (быстродействие, удобство пользования, минимум действий на выполнение операций). Например – интерфейс кассира в супермаркете. Другая крайность – сценарии, выполняемые от случая к случаю, не каждый день и не требующие особой оперативности (например, расчёт заработной платы за месяц). Эти данные позволяют, структурировать подачу информации, убрать из «главных» интерфейсов редко используемые опции и т.п.

Пример. Фрагмент описания потока событий ИСП для прецедента «Оформить заказ для нового клиента», расширенного объёмами и средними значениями объектов (текст в круглых скобках).

В процессе выполнения прецедента менеджер по приёму заказов выбирает заказчика из клиентской базы (в 95% случаев), либо вызывается интерфейс регистрации нового клиента (в 5% случаев).

Литература к лекции

14. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.:Мир, 1999. – 191 с., ил.
15. Леоненков. Самоучитель UML.
16. Мацяшек Лешек, А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер. с англ. - М.: Издательский дом "Вильямс", 2002. - 432 с.: ил. - Парал. тит. Англ.
17. Маклаков С.В. Bpwin Erwin Case-средства разработки информационных систем. – Москва “ДиалогМифи ” – 2000
18. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.
19. Орлов С. Технологии разработки программного обеспечения: Учебник. – СПб.: Питер, 2002. – 464 с.: ил.

20. Каменова, Громов. Моделирование бизнеса. Методология ARIS. — М.: Весть-МетаТехнология, 2001.
21. Брауде Э. Технологии разработки программного обеспечения. – СПб: Питер, 2004. – 655 с.: ил.
22. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД “Вильямс”, 2002.
23. Л.Новиков. Введение в Rational Unified Process. http://www.interface.ru/rational/interface/151199/rup/main.htm

6. Документирование и проверка требований

План лекции

Документирование требований в соответствие с ГОСТ РФ
Структура ТЗ в соответствие с ГОСТ 34.602-89
Описание требований к системе в соответствие с ГОСТ 34.602-89
Документирование требований в RUP
Документирование требований на основе IEEE Standard 830-1998
Документирование требований в MSF
Верификация и валидация.
Двусмысленность требований
"Золочение" продукта
Минимальная спецификаций
Пропуск типов пользователей
Методы и средства проверки требований
Неофициальные просмотры требований
Инспекции
Разработка тестов
Определение критериев приемлемости

Чтобы требования, выявленные и описанные (см. материалы лекции 3 и лекции 4) приняли силу соглашения между Заказчиком и Разработчиком, их необходимо оформить в виде документа. В российской практике для этого обычно используется документ «Техническое задание», ТЗ, в западной – «Software Requirements Specification», SRS (спецификация программных требований). По сути это – один и тот же документ, поэтому далее по тексту будем употреблять термин «ТЗ», рассматривая различные шаблоны его построения – как на основе российских ГОСТ, так и западных методологий и стандартов.

Документирование требований в соответствие с ГОСТ РФ

Документирование требований регламентировано российскими ГОСТ 19.201-78 «Техническое задание, требования к содержанию и оформлению» и ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы» (ТЗ на АС) [27-28].

Второй документ, по сути, является более проработанной версией первого, адаптированной к созданию автоматизированных информационных систем, поэтому далее рассмотрена структура ТЗ в соответствие с ГОСТ 34.602-89.

Несмотря на то, что для сферы IT 17 лет – это целая эпоха, данный документ практически не устарел: его авторам удалось разработать сбалансированные рекомендации, абстрагируясь от конкретных технических и технологических решений. Кроме того, он по-прежнему играет роль государственного стандарта РФ и при заключении контрактов с государственными предприятиями Разработчика могут обязать оформить ТЗ в соответствии с духом и буквой этого документа.

Структура ТЗ в соответствие с ГОСТ 34.602-89

В задачи лекции не входит перечисление всех правил и рекомендаций данного ГОСТ, желающие могут ознакомиться с ним непосредственно. Ниже будут перечислены разделы, предусмотренные ГОСТ и рассмотрены основные моменты, на которые следует обратить внимание.

Общие сведения – в этом разделе, помимо юридических реквизитов сторон и пр. деловой информации ГОСТ рекомендует указать источники и порядок финансирования работ.

Назначение и цели создания (развития) системы – здесь необходимо указать показатели объекта автоматизации, которые должны быть достигнуты и критерии оценки достижения этих показателей. Данным разделом на практике часто пренебрегают и совершенно напрасно – ведь именно в этом разделе закладываются высокоуровневые бизнес-требования и формулируются критерии их достижения.

Характеристика объектов автоматизации – достаточно важный раздел. Его основные «разрезы» - организационная структура, структура управления, структура расположения предприятия и его филиалов. Хорошее описание объекта автоматизации позволяет сэкономить время на определение классов пользователей, для крупных территориально-распределённых систем – заложить структуру и топологию сетевых коммуникаций.

Требования к системе – ключевой раздел настоящего документа, поэтому он будет рассмотрен ниже, более подробно.

Раздел **«Состав и содержание работ по созданию системы»**, говоря современным языком, описывает процесс создания системы, включая выбор методологии, определяющий содержание стадий, этапов и фаз и его конкретизацию для проекта (количество этапов и итераций, их основное содержание).

Порядок контроля и приемки системы – также один из ключевых компонент ТЗ. Он распределяет роли Заказчика и Разработчика в подготовке системы к испытаниям и проведению испытаний. Здесь уместно оговорить правила проведения испытаний, сформулировать основные тестовые сценарии и критерии приёмки.

Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие, опять же, аппелируя к современной терминологии, оговаривают порядок проведения реинжиниринга предприятия, который необходимо осуществить для того, чтобы добиться от внедрения АИС должного эффекта (подбор и обучение персонала, изменения в организационной структуре и т.п.).

Документ заканчивается разделами «**требования к документированию**» и «**источники разработки**», определяющими, соответственно, перечень и формы документации, подлежащей разработке и перечень уже имеющихся документов, содержащих предпосылки для разработки.

В качестве приложений ГОСТ рекомендует использовать **расчет ожидаемой эффективности системы** и **оценку научно-технического уровня системы**.

Описание требований к системе в соответствие с ГОСТ 34.602-89

ГОСТ разделяет все **требования к системе** на три класса:

- требования к системе в целом;
- требования к функциям (задачам), выполняемым системой;
- требования к видам обеспечения.

Среди требований *к системе в целом* (системные требования) указываются требования к:

- структуре системы (здесь закладываются высокоуровневые архитектурные решения, либо структурные ограничения, вводится деление на подсистемы, комплексы и модули, решаются вопросы коммуникации компонент системы и системы с внешним миром),
- режимам функционирования системы;
- персоналу (указывается численность, требуемая квалификация и режим работы);
- надежности;
- безопасности;
- эргономике и технической эстетике;
- транспортабельности для подвижных АС;

- эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы;
- защите информации от несанкционированного доступа;
- сохранности информации при авариях;
- защите от влияния внешних воздействий;
- патентной чистоте;
- стандартизации и унификации,

а также показатели назначения (параметры, характеризующие степень соответствия системы ее назначению) и дополнительные требования (распространяются на обучающие подсистемы, средства контроля работоспособности системы и др.).

Требования ГОСТ *к функциям (задачам)*, в переводе на современный язык, подразделяются на:

- перечень функциональных требований в привязке к подсистемам и очередям автоматизации;
- временной регламент реализации функциональных требований;
- требования к качеству реализации каждого из функциональных требований (в том числе - форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов);
- перечень и критерии отказов для каждого функционального требования, по которому были заданы требования по надежности.

Требования к *видам обеспечения*. Среди видов обеспечения ГОСТ указывает математическое, информационное, лингвистическое, программное, техническое, метрологическое, организационное, методическое.

Документирование требований в RUP

Шаблон SRS, предложенный в RUP[18], по сути представляет собой контейнер, в который необходимо «упаковать» артефакты, полученные в процессе специфицирования требований. Кроме того, SRS частично перекликается с документом «Видение» (см. материалы «лекции 4»). Шаблон удобен своей компактностью и лаконизмом.

1. Введение.
   1.1. *Цель.* Документ должен исчерпывающим образом описывать внешнее поведение системы, а также нефункциональные требования и ограничения.
   1.2. *Краткая сводка возможностей*.
   1.3. *Определения, акронимы и сокращения*.
   1.4. *Ссылки*.
   1.5. *Краткое содержание.*.
2. Обзор системы
   2.1. *Обзор прецедентов*. Содержит список имён и кратких описаний вариантов использования и акторов с иллюстрациями в виде диаграмм прецедентов.
   2.2. *Предположения и зависимости*. Данная секция описывает ключевые технические возможности, компоненты, подсистемы, связанные проекты, которые могут влиять на жизнеспособность разрабатываемой системы.
   Предположением (assumption) называется положение, которое считается истинным при отсутствии доказательства или определяющей информации. [8].
   При определении зависимостей (dependencies) проекта от внешних факторов, необходимо проанализировать, какие новые операционные системы, регламенты бизнес-процессов, стандарты качества, информационные системы

[18] http://www-306.ibm.com/software/rational/

могут появиться на предприятии внедрения и как это может повлиять на функционирование изготовляемой АИС.

3. Описание требований

3.1. *Описание вариантов использования*. Параграф содержит описание вариантов использования и связанных с ними нефункциональные требований, либо ссылки на соответствующие артефакты.

3.2. *Специальные требования*. Параграф содержит описание функциональных требований (не описанных, как варианты использования), а также описание нефункциональных требований общего характера (не сопоставленных ни одному прецеденту в предыдущем разделе), либо ссылки на соответствующие артефакты.

4. Вспомогательная информация. Сюда включается информация, облегчающая понимание документа. Это может быть оглавление и приложения, например, описывающие прототипы пользовательского интерфейса.

Документирование требований на основе IEEE Standard 830-1998

Рассмотрим шаблон документа описания требований, составленный К.Вигерсом [8] на основе стандарта [25]. Данный стандарт содержит развёрнутое описание требований, которое может быть оптимизировано для нужд конкретной организации.

1. Введение

1.1 Назначение документа.

1.2. Поддерживаемые соглашения.

1.3. Предполагаемая аудитория и рекомендации по последовательности работы с документом для каждого класса читателей.

1.4. Границы проекта. Здесь содержится ссылка на документ «Концепция», если таковой имеется, либо краткое резюме продукта.

1.5. Ссылки.

2. Общее описание.

2.1. Общий взгляд на продукт. Здесь необходимо определить - является ли описываемый продукт новым членом растущего семейства продуктов, новой версией существующей системы, заменой существующего приложения или совершенно новым продуктом. Если спецификация требований определяет компонент более крупной системы, укажите, как это ПО соотносится со всей системой и определите основные интерфейсы между ними.

2.2. Особенности продукта. Перечисляются ключевые особенности продукта или его главные свойства. Здесь уместно поместить контекстную диаграмму (в виде диаграммы вариантов использования, потоков данных или др. спецификаций).

2.3. Классы и характеристики пользователей. Документируется процесс поиска акторов, в котором выявляются все пользователи системы и осуществляется обобщение (выделение классов) пользователей. Найденные классы описываются (например – уровень квалификации, доступный функционал и т.д.).

2.4. Операционная среда. Рассматривается среда функционирования АИС, включая аппаратные средства, операционные системы, для распределённых систем – географическое расположение пользователей и серверов, топология сети.

2.5. Ограничения проектирования и реализации. Рассмотрим классификацию ограничений [8]:

- определенные технологии, средства, языки программирования и базы данных, которые следует использовать или избегать;
- ограничения, налагаемые операционной средой продукта;

- обязательные соглашения или стандарты разработки;
- обратная совместимость с продуктами, выпущенными ранее;
- ограничения, налагаемые бизнес-правилами;
- ограничения, связанные с оборудованием, например требования к быстродействию, ограничения памяти или процессора;
- соглашения, связанные с пользовательским интерфейсом существующего продукта, которые необходимо соблюдать при его улучшении
- форматы и протоколы обмена данными.

2.6 Документация для пользователей.

2.7 Предположения и зависимости

3. Функции системы

Для каждой i-й функции составляется следующее описание.

3.i Наименование i-й функции системы.

3.i.1 Описание и приоритеты. Приводится краткое описание функции и указывается её приоритет (степень важности/очерёдности реализации).

3.i.2 Последовательности «воздействие - реакция». Необходимо перечислить последовательность воздействий, оказываемых на систему (действия пользователей, сигналы внешних устройств и др.), и отклики системы, определяющие реакцию конкретной функции.

3.i.3 Функциональные требования. Необходимо дать детализацию i-й функции, перечислить детализированные функциональные требования, включая реакцию на ожидаемые ошибки и неверные действия. Каждому детальному функциональному требованию присваивается уникальный идентификатор.

4. Требования к внешнему интерфейсу

Ниже рассмотрены конкретные рекомендации по написанию разделов этого параграфа, согласно [8]:

4.1 Интерфейсы пользователя

Основные характеристики UI:

- ссылки на стандарты графического интерфейса пользователей или стилевые рекомендации для семейства продукта, которые необходимо соблюдать;
- стандарты шрифтов, значков, названий кнопок, изображений, цветовых схем, последовательностей полей вкладок, часто используемых элементов управления и т.п.;
- конфигурация экрана или ограничения разрешения;
- стандартные кнопки, функции или ссылки перемещения, одинаковые для всех экранов, например кнопка справки;
- быстрые клавиши;
- стандарты отображения сообщений;
- стандарты конфигурации для упрощения локализации ПО;
- специальные возможности для пользователей с проблемами со зрением.

4.2 Интерфейсы оборудования

Опишите характеристики каждого интерфейса между компонентами ПО и оборудования системы. В описание могут входить типы поддерживаемых устройств, взаимодействия данных и элементов управлений между ПО и оборудованием, а также протоколы взаимодействия, которые будут использоваться,

4.3 Интерфейсы ПО

Опишите соединения продукта и других компонентов ПО (идентифицированные по имени и версии), в том числе базы данных, операционные системы, средства, библиотеки и интегрированные коммерческие компоненты. Укажите назначение элементов сообщений, данных и элементов управления, обмен которыми происходит между компонентами ПО. Опишите службы, необходимые внешним компонентам ПО, и природу взаимодействия между компонентами. Определите данные, к которым будут иметь доступ компоненты ПО.

4.4 Интерфейсы передачи информации

Укажите требования для любых функций взаимодействия, которые будут использоваться продуктом, включая электронную почту, Web-браузер, протоколы сетевого соединения и электронные формы. Определите соответствующие форматы сообщений. Опишите особенности безопасности взаимодействия или шифрования, частоты передачи данных и механизмов синхронизации.

5. Другие нефункциональные требования

В этом разделе описываются остальные нефункциональные требования, не относящиеся к требованиям к интерфейсу, которые представлены в разделе 4, и к ограничениям, описываемым в разделе 2.5.

5.1 Требования к производительности

Укажите специальные требования к производительности для различных системных операций. Обоснуйте их необходимость для того, чтобы помочь разработчикам принять правильные решения, касающиеся дизайна. Например, из-за жестких требований к времени отклика базы данных разработчики могут зеркализовать базу данных в нескольких географических метаположениях или денормализовать связанные таблиц баз данных для получения более быстрого ответа на запрос.

Приложение А. Словарь терминов (глоссарий).

Приложение Б. Модели анализа. В этот раздел помещаются все модели, построенные в процессе анализа требований (см. материалы лекции 09-Моделирование требований).

Приложение В. Список вопросов

Это динамический список еще не разрешенных проблем, связанных с требованиями. Это могут быть элементы, помеченные как «TBD» (to be determined — необходимо определить), отложенные решения, необходимая информация, неразрешенные конфликты и т.п.

Документирование требований в MSF

В начале фазы проектирования проектная группа работает с проектными требованиями. Они подразделяются на:

- бизнес-требования,
- требования к эксплуатации,
- системные требования,
- требования пользователя.

Одним из основных результатов фазы проектирования являются

- функциональная спецификация,

которая служит:

- инструкцией команде разработчиков о том, что они должны будут создать;
- основой для оценивания объема работы;
- четкое соглашение с Заказчиком о том, что должно быть сделано;
- основой для синхронизации работы всей проектной команды.

С шаблонами соответствующих документов можно ознакомиться на сайте Microsoft, [26].

Верификация и валидация

Термин «верификация» (verification) в русскоязычной литературе обычно переводят, как «проверка». Термин «валидация» - как «проверка правильности», «аттестация», «утверждение».

Согласно стандарту IEEE 1012-1986, *верификация* представляет собой процесс оценивания системы или компонента с целью определить, удовлетворяют ли результаты некой фазы условиям, наложенным в начале данной фазы. *Валидация* в этом же стандарте определяется, как процесс оценивания системы или компонента во время или по окончании процесса разработки с целью определить, удовлетворяет ли она указанным требованиям.

Трудно ожидать от читателя, впервые столкнувшегося с этой терминологией и её определениями в этом и других стандартах, ясности понимания. По крайней мере, у автора настоящего курса лекций при первом знакомстве с определениями тех же понятий в ISO IEC 12207 возникло чёткое ощущение, что авторы стандарта явно чего-то не договаривают. Посудите сами: согласно данному стандарту, *верификация* – это подтверждение экспертизой и представлением объективных доказательств того, что конкретные требования полностью реализованы. С другой стороны, *валидация* - это подтверждение экспертизой и представлением объективных доказательств того, что конкретные требования к конкретным объектам полностью реализованы. Правда, к чести авторов последнего стандарта, они приводят примечание, которое несколько приближает читателя к пониманию: «валидация связана с экспертизой продукта в целях определения его соответствия *потребностям* пользователя». В этом и заключается суть отличия: если верификация связана с выяснением того, удовлетворяет ли разрабатываемый объект, либо процесс его создания *сформулированным требованиям*, то валидация отвечает на вопрос – правильно ли разработан целевой объект (продукт), удовлетворяет ли он *потребностям* заказчика. Другой аспект валидации заключается в том, что она обычно увязывается с формальной приёмкой (аттестацией) системы.

Некоторые стандарты, например SWEBOK, IEEE 1059-93 “IEEE Guide for Software Verification and Validation Plans”, вводят для этих двух процессов обобщающее понятие V&V (Validation and Verification). Согласно IEEE 1059-93, верификация и валидация программного обеспечения – упорядоченный подход в оценке программных продуктов, применяемый на протяжении всего жизненного цикла. Усилия, прилагаемые в рамках работ по верификации и валидации, направлены на обеспечение качества как неотъемлемой характеристики программного обеспечения и удовлетворение пользовательских требований (конец цитаты).

Из вышесказанного ясно, как осуществить верификацию и валидацию АИС и (или) процесса её создания: в первом случае необходимо убедиться, что АИС (компонента, процесс) соответствует сформулированным требованиям, во втором – что АИС действительно работает! Но если критерием проверки АИС служат требования, то что может послужить критерием проверки самих требований? Ответ заключается в том, что требования должны удовлетворять свойствам, сформулированным в лекции 2, Кроме того, следует убедиться в том, что [8]:

- в спецификации требований к ПО должным образом описаны предполагаемые возможности и характеристики системы, которые удовлетворят потребности различных заинтересованных в проекте лиц;
- требования к ПО точно отражают системные требования, бизнес-правила и др.;
- требования обеспечивают качественную основу для проектирования и сборки ПО.

Некоторые типичные проблемные ситуации процесса формирования и оценки требований

Двусмысленность требований

В ряду проблем и недостатков требований двусмысленность, является, пожалуй, наиболее критичным фактором риска проекта, закладываемого в фазе формирования требований. Двусмысленность (несоответствие свойству ясности, определённости) закладывает под проект «бомбу замедленного действия». На практике требование, сформулированное двусмысленным образом, может привести к различным его интерпретациям представителями Разработчика и Заказчика. Разработчик, руководствуясь своей интерпретацией, определит на её основе архитектурную основу, создаст аналитические и проектные модели и в конечном итоге создаст программный код. Как показывают исследования, цена исправления ошибки вырастает примерно на порядок при переходе между рабочими потоками (от анализа требований к проектированию, от проектирования к реализации и т.д.).

Тем самым, если не заложить средства на проверку требований на предмет двусмысленности в момент их формирования – существует риск непринятия готовой системы в момент приёмо-сдаточных испытаний, т.к. каждая из сторон будет придерживаться своей версией интерпретации требований, что ведёт к убыткам, судебным процессам и т.п. и тому есть масса примеров.

«Золочение» продукта

Под «золочением» [8] понимают такие ситуации, когда разработчики добавляют функции, которых нет в спецификации, но им кажется, что это понравится пользователям. Зачастую же клиентам не нужны такие избыточные возможности, получается, что время, отведенное на реализацию, тратится впустую (конец цитаты).

Эта ситуация возникает в случае, когда, во-первых, в коллективе Разработчика присутствуют творческие личности (ведь далеко не всякая команда станет проявлять инициативу и делать сверх того, о чём её просили), во вторых – существует разрыв в прохождении информации от Заказчика к Разработчику. Инициативный разработчик «золотит» продукт из самых лучших побуждений, но, возможно, он плохо знаком с бизнес-процессом Заказчика и заложенные им «фичи» попросту не будут востребованы.

Другая сторона «золочения» заключается в том, что группа представителей Заказчика неоднородна по своей структуре и может возникнуть ситуация, когда представитель Заказчика, формулирующий «дорогие» требования, не обладает соответствующими полномочиями. Это – специфика российских предприятий, где часто всё бывает устроено существенно неформально.

Минимальная спецификация

Создавать полную документацию требований в соответствии с вышеизложенными принципами, или ограничиться наброском требований на 2-3 страницы, как это зачастую делает автор лекций в небольших проектах – как говориться, дело вкуса.

Однако, для работы «не по правилам», во-первых, должны быть объективные предпосылки, во-вторых – следует отдавать себе отчёт в выгодах и рисках этого выбора.

Минимальная спецификация уместна, если имеет место наличие одновременно трёх обстоятельств (объединение по «И»):

а) цена контракта и размеры проекта таковы, что разработка развёрнутого ТЗ экономически нецелесообразна;

б) коллектив Разработчика обладает достаточной степенью профессионализма и опыта выполнения проектов в смежных областях, чтобы уметь создавать по краткой спецификации продукт, который пройдёт приёмку Заказчиком;

в) между Заказчиком и Разработчиком существуют конструктивные отношения и обе стороны понимают и принимают риски мини-спецификации.

Другой вариант работы по мини-спецификации: Заказчик и Разработчик понимают, что создание развёрнутой спецификации оттягивает окончание выпуска готового продукта, что главная цель проекта – продукт, а не документация и готовы к плотному сотрудничеству в процессе его создания. Это – путь так называемых agile-методологий разработки ПО, подробнее см. в http://www.agile.org.

Основной риск применения мини-спецификации заключаются в том, что они базируются на человеческом факторе. Хорошие и конструктивные отношения между сторонами «на берегу» должны сохраниться на всём протяжении проекта, в противном случае у сторон возникнут существенные проблемы в формальном доказательстве того – что должна делать программа, т.к. мини-спецификация для этого недостаточно полна.

Пропуск типов пользователей

Корпоративные АИС создаются для того, чтобы быть использованными различными группами пользователей. Может сложиться ситуация, в которой в группу представителей Заказчика, участвующих в формировании требований, попадут наиболее инициативные персоны предприятия, которые, по всей видимости, смогут донести свой голос до представителей Разработчика. Те же категории пользователей, у которых не найдётся активных представителей, могут оказаться «за бортом» автоматизации. Именно эта ошибка формирования требований называется «пропуск классов пользователей». Чтобы её избежать, представитель Разработчика должен объективно оценить организационную структуру предприятия и его бизнес-процессы и вдумчиво подойти к выбору ключевых персон, проведение интервью с которыми поможет сформировать целостную картину требований к создаваемой АИС.

Методы и средства проверки требований

Наработано значительное количество методов и средств проверки требований [8,21-31]. Они разнятся по ряду параметров. Так, различают:

- по широте анализа – просмотр (выборочная проверка) и сквозной контроль (тотальная проверка);
- по степени формализации – неофициальные процедуры, процедуры, проводимые по формальным правилам (инспекции, экспертизы);
- по составу группы проверки – с (без) участием автора, с (без) участием менеджера проекта, с (без) участием представителей внешних организаций;
- по используемым средствам – тексты требований, тестовые сценарии, критерии приемлемости, прототипы.

Понятие и методы прототипирования были рассмотрены в лекции 5 Некоторые другие, наиболее важные из перечисленного выше, методы и средства, рассмотрены далее по тексту.

Неофициальные просмотры требований

Различают [8] несколько способов неофициальных просмотров требований:

- просмотр «за столом»,
- коллективная проверка,
- критический анализ.

В первых двух случаях автор требований обращается за помощью к коллегам (соответственно, к одному, либо к нескольким) с целью выдачи практических рекомендаций по улучшению продукта. В третьем случае автор осуществляет презентацию разработанные им требования на совещании с последующим обсуждением.

Неофициальные просмотры используют для знакомства с разработкой, сбора отзывов, формирования обратной связи. По статистике, приведённой в [31], неофициальные просмотры позволяют выявить до 60% ошибок в требованиях.

Инспекции

Понятие инспекции, применительно к IT-индустрии, впервые было сформулировано Майклом Фэганом (Michael Pagan) из IBM в середине 70-х гг.[19].

Согласно стандарту IEEE[20], проведение инспекций, в отличие от неформальных просмотров, базируется на своде формальных требований и правил. Представленный ниже обзор правил приведён, основываясь на работе [6]. Кроме того, слушателям следует порекомендовать ознакомиться с параграфом «Проведение экспертизы» главы 15 монографии [8], где представлено детальное описание процедуры экспертизы.

Лица, занимающие управленческие позиции (менеджеры) в отношении к любым членам команды инспектирования, не должны участвовать в инспекциях.

Инспекция должна вестись под руководством непредвзятого (независимого от проекта и его целей) лидера, обученного техникам инспектирования.

Инспектирование всегда вовлекает авторов промежуточного или конечного продукта.

В группу инспекции входят лидер, регистратор, рецензент и несколько (от 2 до 5) инспекторов. Члены команды инспектирования могут специализироваться в различных областями экспертизы (обладать различными областями компетенции), например, предметной области, методах проектирования, языке и т.п. В заданный момент (промежуток) времени инспекции проводятся в отношении отдельного небольшого фрагмента продукта (в большинстве случаев, фокусируясь на отдельных функциональных или других характеристиках; часто, отталкиваясь от отдельных бизнес-правил, функциональных требований или атрибутов качества, прим. автора). Каждый член команды должен исследовать оцениваемый продукт и другие входные данные до проведения инспекционной встречи, применяя, возможно, те или иные аналитические техники в небольшим фрагментам продукта или к продукту, в целом, рассматривая в последнем случае только один его аспект, например, интерфейсы. Любая найденная аномалия должна документироваться, а информация передаваться лидеру инспекции. В процессе инспекции лидер руководит сессией и проверяет, что все подготовились к инспектированию. Общим инструментом, используемым при инспектировании, является проверочный лист (checklist), содержащий аномалии и вопросы, связанные с аспектами, вызывающими интерес. Результирующий лист часто классифицирует аномалии и оценивается командой с точки зрения его завершенности и точности. Решение о завершении инспекции принимается в соответствии с одним (любым) из трех критериев:

1. Принятие с отсутствием либо малой необходимостью переработки
2. Принятие с проверкой переработанных фрагментов
3. Необходимость повторной инспекции.

Разработка тестов

Механизм вариантов использования (uses cases), рассмотренный в лекции 4, позволяет ответить на вопрос: как будет использоваться система. Чтобы проверить систему, используется аналогичный механизм: тестовых сценариев (test cases).

Тестовые сценарии (ТС) рекомендуется создавать уже на ранних стадиях работы с требованиями, в идеале – после получения запросов совладельцев, параллельно с разработкой вариантов использования.

---

[19] Pagan, Michael E. 1976. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal 15(3): 182-21.

[20] IEEE 1028-97 “IEEE Standard for Software Reviews“

Тестовые сценарии, как и варианты использования, могут поддерживать разные уровни абстракции. Различаются концептуальные и детальные ТС. Концептуальный уровень предполагает проработку процедуры тестирования, инвариантную к конкретной реализации UI.

Как использовать тестовые сценарии для тестирования требований? В [8] предлагается следующая процедура.

1. Построить матрицу, где по вертикали отмечены функциональные требования, а по горизонтали – тестовые сценарии.

2. Убедиться, что каждый из ТС осуществим на существующем наборе требований.

3. Убедиться, что для каждого требования представлен как минимум один ТС.

4. Прочертить «путь» каждого из ТС на карте диалогов. Это позволит: обнаружить некорректные или пропущенные требования, исправить ошибки на карте диалогов и отшлифовать варианты тестирования.

Как быть с тестированием нефункциональных требований? Согласно [33], процедура анализа требований считается выполненной только тогда, когда все требования, включенные в спецификацию, обладают методами оценки соответствия им создаваемого программного продукта.

Для того, чтобы нефункциональные требования были измеримы, каждому из них в идеале необходимо сопоставить количественную метрику. Если это не удаётся – возможно, требование следует переформулировать, либо детализировать.

Определение критериев приемлемости

При формальной приёмке продукта существуют две типовые процедуры: демонстрация продукта Разработчиком на тестовых сценариях и проверка продукта Заказчиком. Далеко не каждого Заказчика можно убедить, что он не должен «тыкать кнопки», лежащие за пределами тестовых сценариев. Однако, в период стабилизации продукта, для Заказчика важнее даже не количество выявленных дефектов, а возможность проверки – годится ли разработанная АИС для решения поставленных им задач.

Чтобы не откладывать столь важный вопрос до момента приёмки системы, крайне важно, наряду с формированием требований, вовлечь Заказчика на ранних стадиях создания продукта в процесс формирования *критериев приемлемости*. Критерии приемлемости (acceptance criteria)[21] должны отразить точку зрения Заказчика на то, что он считает *правильной системой*.

Делегирование разработки тестов на приемлемость пользователям — эффективная стратегия разработки требований [8]. Это позволяет уже на этапе сбора информации перейти от формулировки вопроса с «Что вам нужно делать с помощью системы?» к «Как вы делаете вывод о том, что система удовлетворяет вашим потребностям?». Если клиент не может описать, как он оценит, что конкретное требование удовлетворено системой, значит, требование сформулировано недостаточно ясно.

Раннее формирование тестов для проверки приемлемости позволяет обнаружить дефекты в требованиях.

Проверка приемлемости базируется на ключевых (существенных) вариантах использования. При этом следует абстрагироваться от альтернативных сценариев и исключений и сосредоточить внимание на основном потоке событий. Необходимо учесть также и нефункциональные требования, такие, как производительностью, легкость и простота использования.

---

[21] IEEE. 1990. IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology. Los Alamitos, CA: IEEE Computer Society PreEis.

Литература к лекции

24. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговыйi дом «Русская Редакция», 2004. —576с.: ил.
25. IEEE Standard 830-1998, «IEEE Recommended Practice for Software Requirements Specifications»
26. Белые страницы MSF. http://www.microsoft.com/rus/msdn/msf
27. ГОСТ 19.201-78 «Техническое задание, требования к содержанию и оформлению»
28. ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы» (ТЗ на АС)
29. Брауде Э. Технологии разработки программного обеспечения. – СПб: Питер, 2004. – 655 с.: ил.
30. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД “Вильямс”, 2002.
31. Соммервилл, Иан. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 624 с.: ил. – Парал. тит. англ.
32. Орлик С. Программная инженерия. Качество программного обеспечения (Software Quality) Copyright © Сергей Орлик, 2004-2005 http://www.sorlik.ru/swebok/3-10-software_engineering_quality.pdf
33. Орлик С., Булуй Ю. Введение в программную инженерию и управление жизненным циклом ПО Программная инженерия. Программные требования. Copyright © Сергей Орлик, 2004-2005. http://www.sorlik.ru/swebok/3-1-software_engineering_requirements.pdf

7. Введение в управление требованиями

План лекции

Принципы и приемы управления требованиями
Базовая версия требований
Процедуры управления требованиями
Контроль версий
Атрибуты требований
Контроль статуса требований
Измерение трудозатрат, необходимых для управления требованиями
Управление изменениями
Управление незапланированным ростом объема
Процесс контроля изменений
Анализ влияния изменения
Трассируемость требований

Пройдя этапы выявления, всестороннего анализа, формализации, спецификации, проверки, требования к АИС приобретаю статус документа. Стороны ставят на документе свои подписи, тем самым, удостоверяя, что именно этот (представленный в SRS) набор требований представляет свод законов, по которому создаётся система. Затем осуществляется проектирование и реализация системы. Готовая АИС передаётся Заказчику, который, совместно с Разработчиком осуществляет её приёмку и ввод в эксплуатацию. Такая схема была заложена в подходе, который известен в литературе, как «каскадный» или «водопадный» [22] (см. рис. 13-1). В этой схеме нет места управлению требованиями, т.к. они статичны, сформулированы в начале проекта и неизменны во времени.

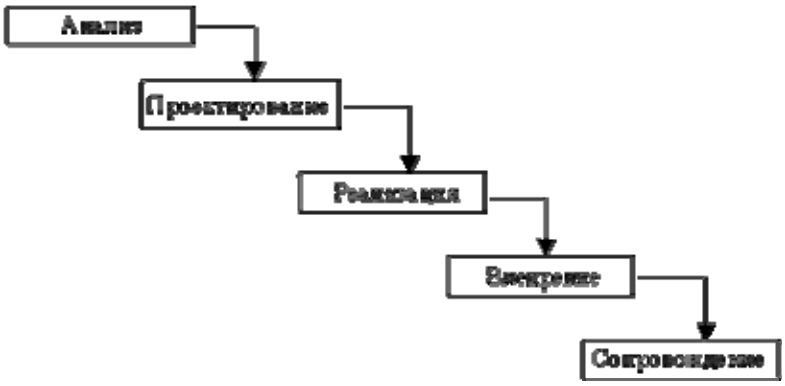


Рис. 13-1.

Каскадный подход представлял собой одну из первых систематизаций потоков работ программной инженерии и на момент своего появления представлял безусловную ценность. Однако практика выполнения проектов автоматизации в рамках данного

[22] Royce, Walker W. Managing the development of large software systems: concepts and techniques. Proc. IEEE WESTCON, Los Angeles, August 1970, pp. 1-9

подхода показала низкий (порядка 20%) процент успешных проектов. Первая причина – лавинообразное разрастание цены исправления ошибок, возникших на ранних этапах создания системы от этапа к этапу (схема не имела обратных связей и, соответственно, ошибки копились вплоть до этапа внедрения). Вторая – статичность схемы. Крупный проект автоматизации может длиться 2, 3 года, а требования, замороженные в SRS, перестают соответствовать бизнес-реалиям предприятия внедрения, которое за столь долгий период может существенно измениться.

Подавляющее большинство современных методологий управления проектами разработки программного обеспечения[23] при всём своём разнообразии сходятся в одном: требования могут меняться! Причём практически на любой фазе производства АИС. Эта новая парадигма работы с требованиями, безусловно, импонирует Заказчикам. Теперь они имеют право ошибиться и исправить свою ошибку. Они могут дать волю своему креативу и постоянно изобретать новые возможности и формы реализации продукта. Но каково Разработчику? Если «двусмысленность – страшилка любой спецификации требований[24]», то неконтролируемое изменение и разрастание требований – ходячий кошмар Разработчика. Вопрос контроля процесса изменений требований и его влияния на другие рабочие потоки программной индустрии настолько серьёзен, что породил отдельную инженерную дисциплину – управление требованиями. Подробно ознакомиться со всеми этапами, артефактами, приёмами и методами данной дисциплины можно, изучив третью главу монографии [8], краткое изложение которой легло в основу этой лекции.

Согласно RUP[25], управление требованиями – это систематический подход к выявлению, организации и документированию требований к системе, а также установка и поддержание соглашения между клиентом и группой разработки по поводу изменений требований к системе. Данное соглашение, как и тексты исходных требований, подлежит документальному оформлению.

Согласно [8], к действиям по управлению требованиями относятся:

- определение основной версии требований (моментальный срез требований для конкретной версии продукта);
- просмотр предлагаемых изменений требований и оценка вероятности воздействия каждого изменения до его принятия;
- включение одобренных изменений требований в проект установленным способом;

---

[23] Исключение составляют, например, некоторые классы военных разработок, где до сих пор применяются модификации каскадного подхода, базирующиеся на очень тщательной проработке и верификации спецификаций и гарантирующие точное соответствие продукта спецификациям.

[24] Lawrence, Brian. 1996. Unresolved Ambiguity. American Programmer 9(5}:17-22

[25] http://www-306.ibm.com/software/rational/

- согласование плана проекта с требованиями;
- обсуждение новых обязательств, основанных на оцененном влиянии изменения требований;
- отслеживание отдельных требований до проектирования, исходного кода и вариантов тестирования;
- отслеживание статуса требований и действий по изменению на протяжении всего проекта.

Принципы и приемы управления требованиями

Базовая версия требований

Чтобы договориться об изменении требований, сначала нужно их зафиксировать в «первозданном виде».

Базовая версия (baseline) – это набор функциональных и нефункциональных требований, которые разработчики обязались реализовать в определенной версии (итерации).

Управление требованиями – это рабочий процесс, следовательно, он должен подчиняться определённым правилам и процедурам.

Процедуры управления требованиями

Процедуры управления требованиями базируются на:

- инструментах, приемах и соглашениях по управлению версиями различных документов требований и отдельных требований;
- правилах составления базовой версии требований;
- статусах требований, которые будут использоваться, и категориях лиц, которые имеют право изменять их;
- процедурах контроля за статусом требования;
- способах, с помощью которых новые требования и изменения существующих требований предлагаются, обрабатываются, обсуждаются и передаются всем заинтересованным лицам;
- методах анализа влияния предложенного изменения;
- отслеживании связей планов и обязательств проекта с изменением требований.

Контроль версий

Каждая версия документа требований должна содержать историю переработки, где указываются внесенные изменения, дата каждого из них, лицо, внесшее изменение, а также причина. Целесообразно добавлять номер версии к названию каждого отдельного требования, который можно последовательно увеличивать при модификации требований.

Для документирования версий используются текстовые процессоры, электронные таблицы. Существуют специализированные средства для контроля версий и конфигураций[26]

Атрибуты требований

С позиций управления, каждое из требований представляет собой самостоятельный объект. Изменения осуществляются в описательной части данного объекта. Контроль изменений удобнее осуществлять с помощью *атрибутов требований*. Набор атрибутов подбирается для каждого проекта индивидуально, исходя из максимальной результативности для команды проекта. При первом внедрении средств управления изменениями рекомендуется использовать не более пяти атрибутов. Это количество можно будет расширить впоследствии, когда команда «войдёт во вкус» процесса управления изменениями и в том случае, если добавление новых атрибутов оправдано практическими соображениями.

В качестве шаблона описания атрибутов требований К.Вигерс [8] предлагает следующий набор:

- дата создания требования;
- номер его текущей версии;
- автор требования;
- лицо, ответственное за удовлетворение требования;
- ответственный за требование или список заинтересованных лиц (чтобы принимать решения о предложенных изменениях);
- состояние требования;
- происхождение или источник требования;
- логическое обоснование требования;
- подсистема (или подсистемы), для которых предназначено требование;
- номер версии продукта, для которого предназначено требование;
- используемый метод проверки или критерий тестирования приемлемости;
- приоритет реализации;
- стабильность требования

Контроль статуса требований

В автоматизированных средствах управления проектами, например MS Project, для контроля степени выполнения той или иной работы используется понятие степени

[26] Брайен А. Уайт. Управление конфигурацией программных средств. Практическое руководство по Rational ClearCase: Пер. с англ. –М.:ДМК Пресс, 2002. – 272 с.: ил. (Серия «Объектно-ориентированные технологии в программировании»).

выполнения (progress), выражаемой в процентах. Данный способ слабо применим в программистских разработках, где, в силу их слабой формализованности, трудно оценить работу в процентах. При управлении требованиями рекомендуется оперировать не процентом, а статусом. К.Вигерс предлагает следующий шаблон для определения статуса требования:

Табл. 13-1.

Состояние	***Определение***
Proposed (Предложено)	Требование запрошено авторизированным источником
Approved (Одобрено)	Требование проанализировано, его влияние на проект просчитано, и оно было размещено в базовой версии определенной версии. Ключевые заинтересованные в проекте лица согласились с этим требованием, а разработчики ПО обязались реализовать его
Implemented (Реализовано)	Код, реализующий требование, разработан, написан и протестирован. Требование отслежено до соответствующих элементов дизайна и кода
Verified (Проверено)	Корректное функционирование реализованного требования подтверждено в соответствующем продукте. Требование отслежено до соответствующих вариантов тестирования. Теперь требование считается завершенным
Deleted (Удалено)	Утвержденное требование удалено из базовой версии. Опишите причины удаления и назовите того, кто принял это решение
Rejected (Отклонено]	Требование предложено, но не запланировано для реализации ни в одной будущих версий. Опишите причины отклонения требования и назовите того, кто принял это решение

Измерение трудозатрат, необходимых для управления требованиями

Управление требованиями, как и всякий другой процесс, требует ресурсов. Контроль усилий также позволяет выяснить, выполняют ли разработчики предполагаемые задачи для управления требованиями. Основные трудозатраты по управлению требованиями [8]:

- предложение изменения требований и новых требований;
- оценка предложенных изменений, включая оценку влияния изменения;
- изменение работы;
- обновление документации требований или базы данных;
- сообщение об изменениях требований заинтересованным группам и отдельным лицам;
- контроль и отчет о состоянии требования;
- сбор информации о трассируемости требований.

Управление изменениями

Управление незапланированным ростом объема

Незапланированный рост объема ставит под удар:

- 80% проектов по разработке систем управления информацией;
- 70% проектов по разработке военных систем ПО;
- 45% проектов по созданию ПО, выполняемых по контракту.

Незапланированным изменением требований считается предложение новой функциональности и существенной модификации после утверждения базовой версии требований к проекту. Чем дольше продолжается работа над проектами, тем больше их объем.

Проблема заключается не в изменении требований, а в том, что запоздалые изменения оказывают существенное влияние на уже проделанную работу. Если каждый запрос на изменение будет удовлетворяться – проект, возможно, никогда не будет завершён.

Ключевая стратегия ограничения роста незапланированных требований – разработка *хороших* требований, руководствуясь приёмами и методами, рассмотренными в предыдущих лекциях, в максимальном контакте с Заказчиком.

Другая стратегия – это умение сказать: «Нет». Психология большинства людей устроена так, что им тяжело отказывать, что в данном случае может привести к состоянию постоянного прессинга. К.Вигерс предлагает «смягчить» этот подход, заменив «Нет», на «Не сейчас» (требование обязательно будет выполнено, но не в текущей версии). Автор лекционного курса, соответственно, хотел бы добавить в эту копилку фразу «Зачем?». Опыт показывает, что данная фраза значительно упрощает общение и позволяет сподвигнуть представителя Заказчика на размышления: а действительно ли его идея хороша, а какую конкретную пользу она принесёт бизнесу его предприятия?

Однако, не следует делать вывод из всего вышесказанного, что изменения не нужны. Изменения неизбежны, приемлемы и в ряде случаев благоприятны. Бизнес-процессы, рыночные возможности, конкурирующие продукты, и технологии — все они могут меняться в ходе разработки продукта, и менеджеры могут определить, как в ответ на эти изменения необходимо откорректировать направление работы.

Процесс контроля изменений

Процесс контроля изменений позволяет руководителю проекта принимать информированное бизнес-решение, которое удовлетворяет клиента и имеет коммерческую ценность, при этом контролируются затраты на жизненный цикл продукта. Вы можете отслеживать статус всех предложенных изменений и убедиться, что предложенные

требования не были потеряны или упущены. После утверждения базовой версии набора требований придерживайтесь этого порядка для внесения всех предлагаемых изменений в нее.

Клиенты и другие заинтересованные лица часто уклоняются от нового процесса, однако процесс контроля изменений — не препятствие для модификации. Это механизм суммирования и фильтрации, дающий уверенность, что в проект включены наиболее ценные изменение.

Если предложенное изменение не настолько важно, чтобы заинтересованное в проекте лицо потратило пару минут на передачу его через стандартные, простые каналы, то стоит задуматься о его ценности вообще. Процесс управления должен быть тщательно задокументирован, максимально прост и, что важнее всего, эффективен.

После регистрации запроса на изменение необходимо принять решение о его дальнейшей судьбе. *Совет по управлению изменениями* (change control board, CCB) (иногда его называют советом по управлению конфигурацией) был признан лучшим практическим решением при разработке ПО[27]. На практике функции Совета могут быть делегированы и одному человеку (в зависимости от размеров проекта).

Запрос на изменение может быть принят, либо отклонён. В первом случае его необходимо включить в план работ над проектом, во втором – сформулировать мотивированный отказ. При принятии решения по запросу необходимо исходить: а) из степени важности запроса для Заказчика и б) из его стоимости для Разработчика. Стоимость определяется на основании *анализа влияния изменения*.

Анализ влияния изменения

Анализ влияния обеспечивает точное понимание подтекста предложенного изменения, что помогает команде принимать информированные бизнес-решения о том, какое изменение одобрить. Анализ позволяет выявить компоненты, которые может понадобиться создать, изменить или отклонить, и оценить затраты, связанные с реализацией изменения. До того, как разработчик ответит: «Конечно, без проблем», он должен потратить время на анализ результата изменения.

Председатель совета по управлению изменениями обычно просит опытного разработчика выполнить анализ результата определенного.

Анализа результатов изменений затрагивает три аспекта [8].

1. Определите возможные последствия изменения. Часто они вызывают значительный волновой эффект. Включение множества функций в продукт может снизить

[27] McConnell, 1996

его производительность до неприемлемого уровня, например, когда системе, запускаемой ежедневно, потребуется 24 часа для завершения одного запуска.

2. Определите все файлы, модели и документы, которые, возможно, придется изменить, если команда включит все запрошенные изменения.

3. Определите задачи, необходимые для реализации изменения, и оцените усилия, необходимые для выполнения этих задач.

Трассируемость требований

Связи трассируемости помогают следить за развитием требования в обоих направлениях— от первоисточника к реализации и наоборот. Трассируемость представляет собой одну из качеств хороших требований, см. материалы лекции 2

Для осуществления анализа трассируемости каждое требование должно быть уникально идентифицировано.

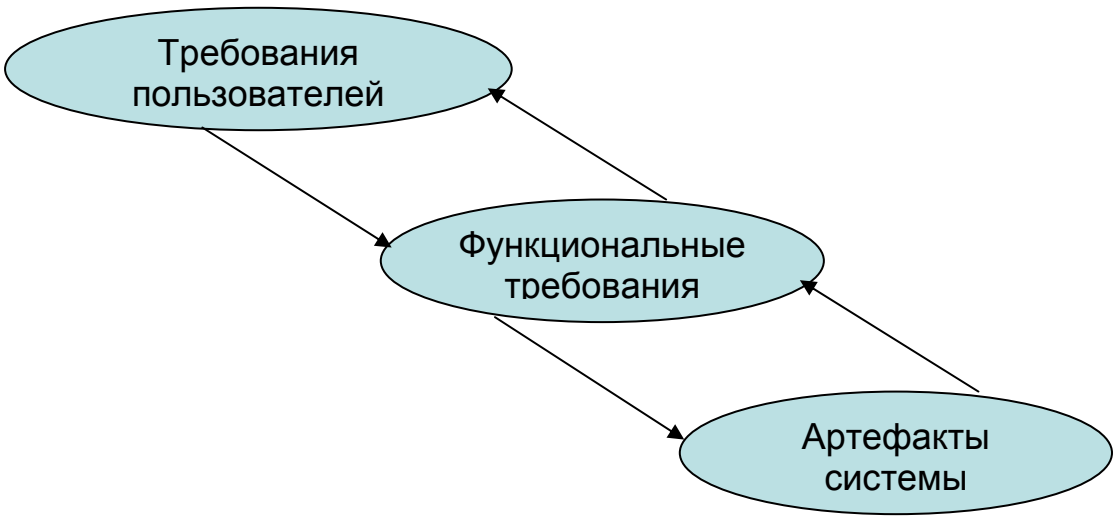


Рис. 13-1. Основные типы трассируемости требований

Требования пользователей отслеживаются в направлении к формально специфицированным функциям системы, чтобы понять, которые требования будут затронуты, если потребности клиентов изменятся. Это также позволяет убедиться в том, что в спецификации требований отражены все потребности клиента. Можно осуществить анализ и в обратном направлении, чтобы определить происхождение каждого требования к ПО.

В процессе анализа, проектирования и реализации компонент системы можно отслеживать связи, ведущие от требований к артефактам (документам, моделям, программным модулям и т.п.) системы. Этот тип связи гарантирует, что каждое требование удовлетворено, поскольку вы знаете, какой компонент соответствует каждому требованию.

Четвертый тип связи контролирует отдельные артефакты в направлении к требованиям для того, чтобы вы знали причину созданию каждого из них.

Предположим, тестеровщик обнаружит незапланированную функциональность при отсутствии соответствующего требования. Этот фрагмент кода может свидетельствовать, что разработчик реализовал официальное требование, которое аналитик теперь может добавить к спецификации. Или же это может быть код-«сирота», украшающий фрагмент, который не относится к продукту. Связи трассируемости помогут вам отсортировать подобные ситуации и получить более полное представление о том, как именно фрагменты вашей системы составляют одно целое. И наоборот, варианты тестирования, которые созданы на основе отдельных требований и которые можно проследить до этих требований, также представляют собой механизм выявления нереализованных требований, поскольку ожидаемой функциональности не будет.

Наиболее типичный способ представления связей между требованиями и другими элементами системами — матрица трассируемости требований, которую также называют матрицей отслеживания требований или таблицей трассируемости (requirements traceability matrix). В табл. 13-2 показана иллюстрация части такой матрицы из [8].

Табл. 13-2.

Пользова-тельское требование	Функциональное требование	Элемент дизайна	Модуль кода	Вариант тестирования
UC-28	catalog.query.sort	Каталог класса	catalog.sort()	search.7 search.8
UC-29	catalog.query.import	Каталог класса	catalog.import() catalog.validate()	search. 12 search. 13 search. 14

Другая форма представления связей трассируемости – дерево трассировок [15].

Литература к лекции

34. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.
35. Л.Новиков. Введение в Rational Unified Process. http://www.interface.ru/rational/interface/151199/rup/main.htm
36. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД “Вильямс”, 2002.
37. Соммервилл, Иан. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 624 с.: ил. – Парал. тит. англ.
38. Орлик С., Булуй Ю. Введение в программную инженерию и управление жизненным циклом ПО Программная инженерия. Программные требования. Copyright © Сергей Орлик, 2004-2005. http://www.sorlik.ru/swebok/3-1-software_engineering_requirements.pdf

8. Совершенствование процессов работы с требованиями

План лекции

Модели совершенствования
ISO9000
SEI-CMM, SEI-CMMI
Принципы совершенствования
Процесс совершенствования
Оценка текущих приёмов
Планирование
Создание и апробация новых процессов
Оценка результатов и приятие решений

Парадигма управления качеством, как способ организации производства, появилась давно. Идеи, заложенные в группе стандартов ISO9000[28], уходят корнями, в частности, и в такие «советские» изобретения, как поддержка рационализаторских предложений, наставничества и др.

В современной концепции процессно-ориентированной организации бизнеса процесс непрерывного совершенствования качества занимает одну из ключевых позиций.

Применительно к софтверной индустрии, помимо серии ISO9000, наиболее успешно себя зарекомендовавшими стандартами качества являются SEI CMM, SEI CMMI, ISO/IEC 15504 (SPICE), Bootstrap, TickIT.

Модели совершенствования

ISO9000

Активное внедрение методов управления качеством на Западе началось в начале 1960-х годов. В основу стандартов серии ISO9000 легла философия подходов CPI (Continuous Process Improvement) и TQM (Total Quality Management) [39]. Подъём экономики послевоенной Японии во многом был обусловлен идеям, заложенным в TQM.

Качество – термин, который для одних означает необходимость делать то, что желает потребитель, для других — то, что отвечает его потребностям. Менеджмент качества, как он определен в ИСО 9001:2000, исходит прежде всего из того, что люди работают лучше, если им известно то, чем они занимаются. [39].

Поэтому прежде, чем приступить к какому-либо действию, следует получить ответы на вопросы: что нужно делать, кто будет проверять сделанное, как это нужно делать и как определить, что работа завершена? Необходимо также продумать, как вы собираетесь управлять данным процессом и как его можно усовершенствовать.

[28] Наиболее распространённые стандарты в области управления качеством

Основные принципы ISO9000:

- Концентрация на потребностях заказчика;
- Активная лидирующая роль руководства;
- Вовлечение исполнителей в процессы совершенствования;
- Реализация процессного подхода;
- Системный подход к управлению;
- Обеспечение непрерывных улучшений;
- Принятие решений на основе фактов;
- Взаимовыгодные отношения с поставщиками.

Безусловное достоинство стандартов этой группы связано с тем, что они апробированы на множестве предприятий мира. Однако рекомендации данных стандартов носят достаточно общий характер и не учитывают специфику предприятий отрасли IT. Для этого были разработаны специализированные подходы, рассмотренные ниже.

SEI-CMM, SEI-CMMI

Стандарт CMM (the Capability Maturity Model) разработан институтом инженерии программного обеспечения (SEI) при университете Карнеги-Меллон.

Назначение стандарта – оценка уровня «зрелости» (maturity levels) организации – разработчика программного обеспечения. Выделяются пять уровней: начальный, повторяемый, определённый, управляемый и оптимизирующий (подробнее см. в [22-8]). Данный стандарт получил широкую известность, значительное количество западных IT-компаний сертифицировано по CMM.

В 2000 г. SEI выпустил CMMI-SE/SW, интегрированную модель совершенствования как ПО, так и возможностей конструирования систем [8].

CMMI-SE/SW имеет две формы. Ступенчатое представление (the staged representation) соответствует структуре SW-CMM с небольшими уточнениями наименований уровней. Пять уровней зрелости содержат 22 области технологических процессов, показанных в таблице 14-1. (CMU/SEI, 2000a). Непрерывное представление (continuous representation), содержит другой взгляд: те же 22 области структурируются по 4 категориям: управление процессами, управление проектами, конструирование и поддержка (CMU/SEI, 2000b).

В непрерывном представлении вместо уровней зрелости определяются шесть уровней способностей (capability levels) для каждой области технологических процессов. Это представление позволяет каждой организации решать, какой уровень способностей ей соответствует в каждой из 22 областей технологических процессов.

Как и в CMM, в рассматриваемом стандарте на уровне 2 имеется область, именуемая «Управление требованиями», но, в отличие от предыдущего стандарта, на уровне 3 есть и отдельная область «Разработка требований». Размещение этой области на уровне 3 не подразумевает, что требования для проектов организации, не достигших уровня 2, собирать и документировать не нужно. Управление требованиями рассматривается как способ, помогающий создавать более предсказуемые и менее хаотичные проекты, что составляет сущность уровня 2 CMM. Приняв порядок управления изменениями и проверки статуса требований, организация может больше внимания уделять разработке высококачественных требований [8].

Табл. 14-1

Уровень зрелости	*Название*	*Области процессов*
1	Начальный	(нет)
2	Управляемый	**Управление требованиями** Планирование проекта Мониторинг и контроль проекта Управление соглашениями с поставщиками Измерения и анализ Обеспечение качества процессов и продуктов Управление конфигурацией
3	Определённый	**Разработка требований** Техническое решение Интеграция продуктов Верификация Валидация Концентрация внимания на процессе Определение процесса организацией Организационное обучение Интегрированное управление проектом Управление риском Анализ и разрешение вопросов
4	Количественно управляемый	Производительность организационных процессов Количественное управление проектом
5	Оптимизирующий	Организационные нововведения и их развёртывание Случайный анализ и разрешение

Область процессов «Управление требованиями». Ключевые темы включают в себя то, как команда разработчиков должна приобретать понимание требований и разрешать вопросы с клиентами, вовлекать участников проекта в работу с требованиями и управлять изменениями. В отличие от SW-CMM, трассирование (одно из ключевых свойств требований) включено в рассматриваемую область процессов. В стандарте обсуждаются следующие качества трассирования:

- обеспечение записи источников низкоуровневых или вторичных требований;
- трассирование каждого требования вниз, к вторичным требованиям, и его размещение по функциям, объектам, процессам и исполнителям;
- установка горизонтальных связей между требованиями, принадлежащими к одному типу.

Область процессов «Разработка требований».

В CMMI-SE/SW описаны три набора приемов разработки требований:

- приемы, определяющие полный набор требований клиентов, которые затем используются для разработки требований для продукта (выявить нужды

заинтересованных в проекте лиц; преобразовать нужды и ограничения в требования клиентов);

- приемы, определяющие полный набор требований для продукта (установить компоненты продукта; разместить требования по компонентам продукта; определить требования к интерфейсу);
- приемы получения вторичных требований, понимания требований и их подтверждения (установить оперативные концепции и сценарии; определить требуемую функциональность системы; проанализировать вторичные требования; оценить стоимость, сроки и риск создания продукта; утвердить требования).

Как в CMM, так и в CMMI формулируются цели, к которым проект или организация по разработке ПО должны стремиться в различных областях процессов. В них также рекомендуются технические приемы, помогающие достичь этих целей.

CMMI-SE/SW регламентирует взаимосвязи между управлением требованиями, разработкой требований и другими областями процессов (рис. 14-1).

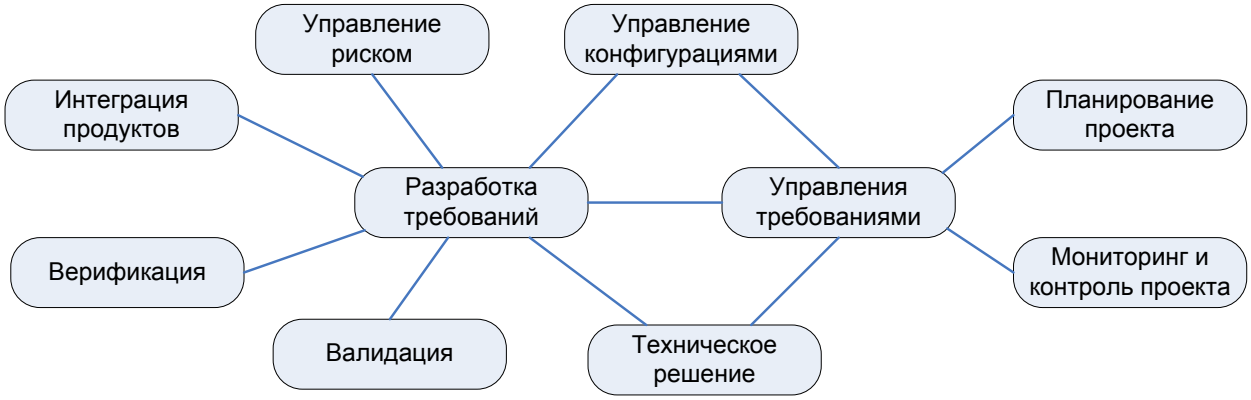


Рис. 14-1

Принципы совершенствования

В [8] сформулированы следующие принципы совершенствования качества программных систем:

- поэтапность,
- непрерывность,
- цикличность,
- наличие стимула,
- ориентация на цели,
- проектный подход.

Мероприятия по совершенствованию процессов следует вводить *поэтапно*. Людям нередко бывает тяжело отказываться от старых привычек, привыкать к новому. Предложенные изменения должны пройти проверку опытом; не всё из предложенного обязательно даст эффект, какие-то новации придётся пересмотреть, от чего-то – отказаться.

Непрерывность – один из ключевых принципов управления качеством: часто мероприятия проводятся в виде кампании, которая затухает после получения сертификата. Организация, которая стремится к лидерству, должна поддерживать «дух качественной работы» постоянно.

Бизнес-процесс улучшения требований характеризуется *цикличностью* (см. Процесс совершенствования): его основные этапы повторяются на всё более высоком уровне. Циклы оптимизации в софтверных организациях удобно приурочивать к проектам, выполняемых в рабочих группах. Анализ недостатков целесообразно производить тогда, когда они в «оперативной памяти» группы проекта, например – один раз в середине проекта и один – сразу после его окончания. Каждый проект по-своему уникален и несёт в себе потенциал для улучшения процессов.

Основным *стимулом* к изменениям К.Вигерс считает трудности, с которыми столкнулась команда проекта, например:

- разработчики не уложились в график, потому что непонятные и неоднозначные требования попали к ним поздно;
- разработчикам пришлось много работать сверхурочно, потому что непонятные или расплывчатые требования были уточнены слишком поздно в процессе разработки;
- попытка тестирования системы не удалась, потому что тестировщики не понимали, что продукт должен делать;
- нужная функциональность была реализована, но пользователи не удовлетворены вялой производительностью, неудобством работы или другими недостатками качества продукта;
- организации пришлось пойти на высокие расходы на сопровождение, потому что клиентам потребовалась масса дополнительных функций, которые следовало определить во время составления требований;
- организация-разработчик ПО приобрела плохую репутацию поставщика продуктов, которые не нравятся клиентам.

Изменения технологических процессов должны быть *целеориентированы*. Примеры целей:

- уменьшение объема работы, вызванного проблемами с требованиями;
- повышение точности планирования и реалистичности планов;
- снижение (исключение) числа ситуаций появления новых требований на финишных этапах проекта.

Действия по совершенствованию процессов должны планироваться, контролироваться и управляться, как *мини-проекты*. Это упрощает выделение требуемых ресурсов, отслеживание перемен и оценки результативности изменений.

Процесс совершенствования

На рис. 14-1 показан типовой цикл совершенствования процессов при создании программного обеспечения [8].

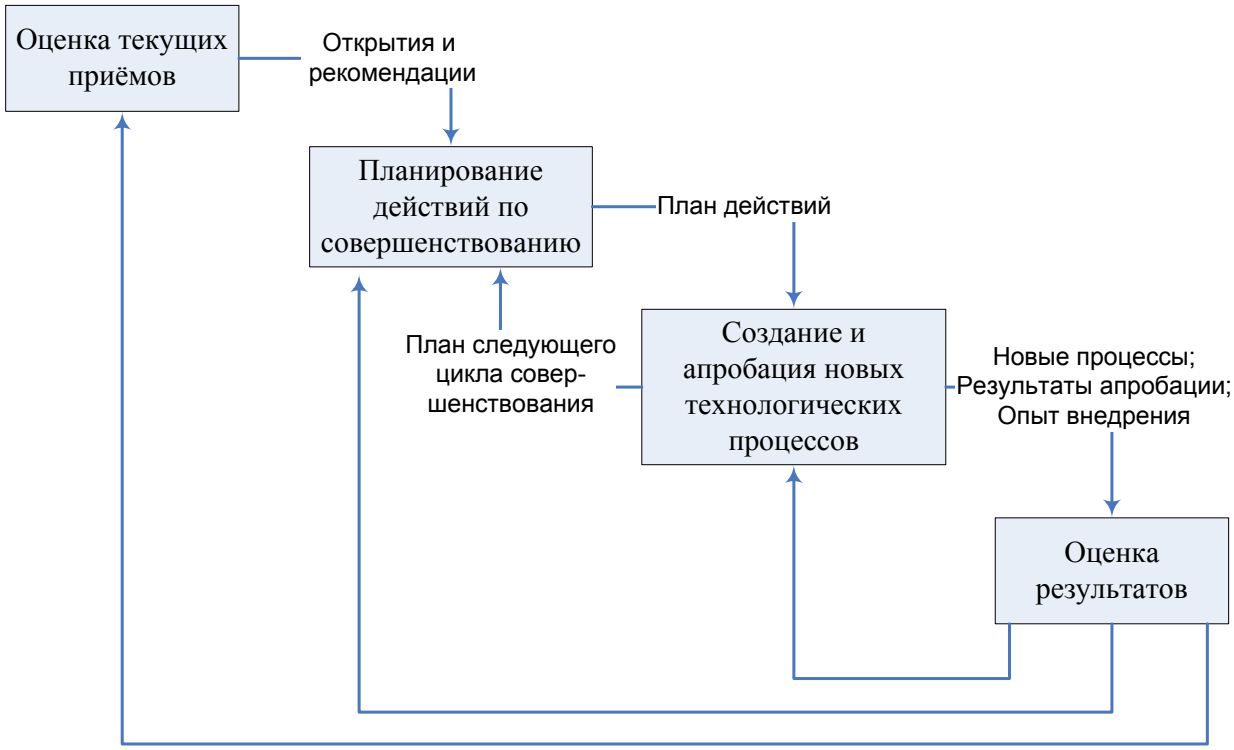


Рис. 14-1.

Оценка текущих приёмов

В соответствие с принципом целенаправленности, в работы по совершенствованию необходимо начать с формулировки целей и оценкой, насколько существующие процессы соответствуют данным целям. Для целей оценки применимы известные в бизнес-моделировании и анализе требований методы и приёмы: от проведения интервью и постановочных семинаров до фиксации модели «Как есть».

Эффективным способом является привлечение внешних консультантов, которые могут составить непредвзятый взгляд на положение в вашей компании.

Результатами оценки является список обнаруженных сильных и слабых сторон в текущих процессах, а также, начальные рекомендации по совершенствованию (переходу к модели «Как надо»).

Планирование

В соответствии с принципом проектного подхода к проведению мероприятий по совершенствованию, для мини-проекта совершенствования необходимо проделать всё то, что обычно делается при инициации проектов: осуществить декомпозицию работ; выделить ресурсы, назначить исполнителей; создать планы работ.

Стратегический план описывает общую программу совершенствования процессов в вашей организации. Тактические планы действий затрагивают конкретные области совершенствования, например процесс сбора требований или процедуру назначения приоритетов [8]. В каждом плане действий должны быть указаны цели действий по совершенствованию, участники и отдельные задачи. План также дает возможность отслеживать выполнение процесса совершенствования, отмечая выполнение отдельных задач.

В плане действий не должно быть более 10 пунктов; срок его реализации не должен превышать 2-3 месяца.

Ниже приведёт шаблон декомпозиции задачи управления требованиями [8].

1. составить проект процедуры управления изменениями;

2. проверить и модифицировать процедуру управления изменениями;

3. провести пробное испытание процедуры управления изменениями

для проекта;

4. модифицировать процедуру управления изменениями на основе обратной реакции по пробному испытанию;

5. оценить инструментальные средства выявления проблем и выбрать одно из них для поддержки процедуры управления изменениями;

6. приобрести выбранное инструментальное средство выявления проблем и настроить его для поддержки конкретной процедуры;

7. внедрить новую процедуру управления изменениями и инструментальное средство в организации.

Создание и апробация новых процессов

Принцип поэтапности призывает не делать «революций» в совершенствовании процессов. Любая новация, описание которой найдено в литературе, заимствована из опыта коллег или разработана лично вами, должна пройти испытание на *вашей команде* и

ваших проектах. Известный неполиткорректный принцип «что русскому хорошо – то немцу смерть» на языке современного менеджмента IT-проектов звучит, как «учёт системы ценностей, принятых в команде разработчиков» [43].

Апробация на реальных задачах – единственный гарантированный способ проверить – годится ли тот или иной инструмент для вашей команды. Чтобы не вовлекать в масштабные эксперименты значительные ресурсы существует способ пилотных (пробных) проектов.

К.Вигерс предлагает следующие методические приёмы при апробации новых процессов:

- выбирайте для участия в пробных проектах людей, которые будут относиться к новым приемам беспристрастно и смогут дать им оценку. Это могут быть как сторонники, так и скептики, но они не должны быть ярыми противниками предлагаемых приемов;
- чтобы результаты было легко истолковать, определите количество критериев, по которым команда будет оценивать пробный проект
- определите заинтересованных лиц, которых следует проинформировать о том, что представляет собой пробный проект и почему он выполняется;
- подумайте о возможности испытания новых процессов по частям в разных пробных проектах. Так вам удастся вовлечь в испытание больше людей, что увеличивает осведомленность, количество откликов и сторонников;
- для более полной оценки поинтересуйтесь у участников пилотных проектов, что бы они почувствовали, если бы им пришлось вернуться к существующим приемам работы.

Оценка результатов и приятие решений.

Оценка результатов апробации новых процессов должна показать – достигнуты ли поставленные цели, стоит ли осуществлять широкомасштабное внедрение новаций, апробированных на пилотном проекте, либо «овчинка выделки не стоит».

Среди ключевых вопросов в области оценки результатов можно выделить следующие [8].

- Насколько гладко прошли пробные проекты и как эффективно они разрешили неопределенности в отношении новых процессов?
- Собираетесь ли вы менять что-либо в следующих пилотных проектах?
- Как прошло общее внедрение новых технологических процессов?
- Удалось ли вам довести до сведения каждого информацию о пользе новых процессов или шаблонов?

- Смогли ли участники понять и эффективно применить новые процессы?
- Собираетесь ли вы менять что-либо при проведении следующего внедрения?

При оценивании результативности достижения поставленных целей следует различать мероприятия, польза от которых проявляется сразу и те, выгода от которых проявится через значительное время. Необходимо учитывать эффект «кривой обучения» (learning curve) [8]: производительность падает, пока люди приспосабливаются к новым способам работы. Кратковременное падение производительности, иногда называемое «лощиной отчаяния» - в — это часть необходимого вклада, который ваша организация вносит в совершенствование процессов.

Невзирая на все возможные трудности, мероприятия по улучшению качества при внимательном к ним отношении неизбежно приведут к повышению эффективности работы команды.

Литература к лекции

39. Калянов Г. Н. Консалтинг при автоматизации предприятий: Научно-практическое издание. Серия «Информатизация России на пороге XXI века». — М.: СИН-ТЕГ, 1997.
40. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД “Вильямс”, 2002.
41. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.
42. Руководство по применению стандарта ИСО 9001:2000 при разработке программного обеспечения/ Пер. с англ. А.Л. Раскина. – М.: РИА “Стандарты и качество”, 2002. – 104 с. – (“Дом качества”, вып. 9 (18)).
43. Коберн А. Быстрая разработка программного обеспечения. – М.: Лори, 2002. 314 с.

9. Требования в управлении проектом. Заключение

План лекции

От рамок проекта к экспресс-планированию
Планирование проекта на основе требований, путь RUP
Требования в гибких методологиях
Артефакты для работы с требованиями в гибких методологиях
Планирование версий и итераций
Анализ требований и управление рисками
Современные тенденции в развитии АИС и технологий их создания
Покупное или заказное ПО - критерии выбора
Стратегии выбора решения
Анализ требований
Анализ несоответствия
Подход на основе лучших практик
Процесс выбора решения

Чтобы определить сметную стоимость и продолжительность работ по проекту автоматизации без грубых ошибок, необходимо выявить и проанализировать требования, а также сформировать архитектурную основу, крайне желательно создать прототипы. И это – как минимум. Иными словами, для того, чтобы создать АИС, сначала нужно проанализировать возможность создания.

Такая постановка вопроса вполне логична и обоснована, это подтвердит любой Разработчик. Однако, она вызывает много вопросов, например:

- Где найти Заказчика, который согласится ждать 2-3 месяца, пока Разработчик составит для него коммерческое предложение?
- Кто оплатит работы по анализу требований? (очевидно, Заказчик)
- Как быть, если цена вопроса окажется непомерной и от проекта придётся отказаться – кто возместит Заказчику убытки на проведение исследований?

Разумный Заказчик сможет найти выход из этого непростого положения, например:

- подыскав Разработчика, обладающего богатым опытом выполнения подобных проектов, который сможет дать требуемую оценку значительно быстрее (но риск ошибки при этом остаётся);
- взяв на себя риски возможного прекращения проекта на ранних стадиях, в случае, если выявится его несоответствие бюджетным ограничениям (в сложных рискованных проектах лучше потерять 5% или 10% от закладываемого бюджета, чем все 100%, как это было в «каскадных» схемах разработки) – путь прогнозирующих методологий
- разделив с Разработчикам ответственность за конечный продукт, приготовившись день за днём работать с ним рука об руку вплоть до появления результата – путь гибких (agile) методологий.

От рамок проекта к экспресс-планированию

Начальную, самую грубую оценку проекта можно сделать на основании документа «Видение». Так, шаблон Vision/Scope MSF содержит список ключевых характеристик/функций, критерии приемлемости и (что очень важно) перечень характеристик/функций, которые лежат «вне рамок» проекта. Параллельно с проработкой концепции, первая фаза MSF содержит работы по анализу рисков: выявляются и оцениваются главные риски проекта.

Чтобы сделать первое приближение плана и сметы проекта на ранних этапах анализа, в [22] предлагается следующий подход:

- Выделить 25 – 99 функций, характеризующих систему (совместно, Заказчик и Разработчик);
- Установить приоритеты для каждой из функций (Заказчик);
- Оценить трудозатраты (Разработчик);
- Оценить риски (Разработчик, возможно привлечение Заказчика);

Все оценки делаются по 3-балльной шкале (высокий, низкий, средний; критический, важный, полезный и т.п.) и сводятся в таблицу:

№ пп.	*Функция*	*Приоритет*	*Трудоёмкость*	*Риск*

Затем, в процессе консультаций Заказчика и Разработчика на основе полученной информации определяется набор функций, который войдут в базовую версию проекта.

Результаты планирования на концептуальном уровне, проделанные, например, на основе вышеуказанного шаблона, позволяют дать первую оценку размерам проекта. Такая оценка не отличается особой точностью, но при определённых условиях (опыт решения Разработчиком аналогичных задач; доверительные отношения между Заказчиком и Разработчиком) может служить отправной точкой для заключения контракта на всю систему.

Планирование проекта на основе требований, путь RUP

RUP поддерживает двухуровневую схему планирования работ над проектом, разделяя план проекта и план итерации. Исходя из базовой концепции планирования итерационных проектов, план проекта разбивается на фазы:

- Начало,
- Уточнение,
- Конструирование,
- Переход.

Исходя из рекомендаций методологии по декомпозиции работ проекта в зависимости от степени сложности проекта и квалификации команды, в каждой фазе выделяется одна или более итераций.

Назначаются даты главных вех (окончания фаз):

- целей жизненного цикла (конец фазы начала, рамки проекта и его бюджет);
- архитектуры жизненного цикла (конец фазы уточнения, законченная архитектура);
- начальной работоспособности (конец фазы конструирования, бета-версия продукта);
- выпуска изделия (конец фазы перехода и полного цикла разработки).

Назначаются даты малых вех, приуроченные к окончанию итераций и их главные цели. Основные цели итераций – выпуск релизов, демонстрируемых, либо передаваемых Заказчику, однако не всякая итерация приводит к выпуску релиза.

План проекта создаётся как можно раньше в начальной фазе и модифицируется по мере необходимости.

Что это означает на практике:

1) укрупнённый план работ составляется «как можно раньше», например – через месяц после начала работ;
2) бюджет появляется лишь к окончанию первой фазы (а она, в зависимости от сложности проекта может длиться месяц, а может и полгода);
3) как план, так и бюджет проекта представляют собой лишь прогноз, который может корректироваться на протяжении работ над проектом;
4) на момент появления плана и бюджета должно появиться подробное описание лишь 20% ключевой функциональности системы и «широкое, но неглубокое» [2] описание 80% функциональности.

Таким образом, концепция укрупнённого планирования в RUP, как типопредставителе класса прогнозирующих методологий, предполагает базировать отношения между Заказчиком и Разработчиком на прогнозах, степень достоверности которых зависит от таких факторов, как качество проработки требований, квалификация команды Разработчика, сложность и новизна проекта.

Более конкретная информация представлена в плане итерации. Основные его особенности:

1) План итерации базируется на функциональных требованиях. К моменту начала итерации совершенно точно известно, какие требования должны быть обработаны (детализированы, спроектированы, запрограммированы) в данной итерации.

План итерации составляется, исходя из сформулированных выше оценок требований – приоритетности, степени риска, трудоёмкости.

2) План итерации имеет жёсткие сроки. В случае проявления незапланированных рисков удовлетворительным вариантом является достижение договорённости о реализации требований данной итерации не в полном объёме, либо переносе требований в следующую итерацию; переносить сроки текущей итерации не рекомендуется;

3) Точный план составляется на одну, очередную итерацию. К моменту окончания текущей итерации должен быть свёрстан план очередной итерации. Такой подход позволяет более гибко работать с рисками.

Таким образом, следует отметить, что требования являются решающим фактором в планировании итерационного проекта.

Требования в гибких методологиях

В противовес прогнозирующим методологиям создания программного обеспечения, относительно недавно сформировалась парадигма гибких (agile) методологий. В феврале 2001 г. инициативная группа из 17 специалистов объединилась в Альянс гибкой разработки программного обеспечения. Эта группа разработала и приняла Манифест гибкой разработки:

- Индивидуальности и взаимодействия ВЫШЕ процессов и инструментов
- Работающее программное обеспечение ВЫШЕ всесторонней документации
- Сотрудничество с клиентами ВЫШЕ переговоров по контракту
- Реакция на изменения ВЫШЕ следования плану

и 12 приложений (в столь же лаконичной форме) к нему[29].

В определённой степени в противовес всему тому, что было сказано в предыдущих лекциях, члены Альянса ставят под сомнение необходимость всестороннего моделирования и документирования требований и даже посягают на святое святых – планы и контракты.

На сегодня «быть гибким» стало модным. Апологеты методологий, заклеймённых членами Альянса, как «прогнозирующие» и даже «тяжеловесные» вступают в дискуссии – можно ли считать адаптировать ту или иную методологию на «гибкие рельсы». Так, опубликованы как минимум два варианта гибкой трансформации для RUP; MSF опубликовало нотацию agile MSF.

Артефакты для работы с требованиями в гибких методологиях

С позиций работы с требованиями основными средствами, которыми оперируют гибкие методологии, являются карты представления системы, истории пользователей, приёмочные тесты и CRC-карты [3-4].

Карта представления в определённой степени заменяет документ «концепция», принятый в прогнозирующих методологиях. В отличие от концепции, представление – это

[29] www.agilealliance.org

текст размером в 20-30 слов, умещающийся на небольшой (размером с визитную) карточке.

Истории пользователей (user story)очень сильно напоминают краткие описания вариантов использования. Особенности историй пользователя – в том, что они, во-первых, должны быть действительно краткими (также умещаться на карточке), во-вторых – в том, что это – действительно истории *пользователей*, т.е. рассказы о том, как они планируют использовать систему. Использование историй пользователя исключает ситуацию, когда аналитик что-то придумал (домыслил) за пользователя – отнюдь, эти артефакты создают сами пользователи. Истории пользователя должны иметь осмысленные наименования и номера.

Истории пользователя, помимо базового функционала, могут содержать *декорации*, очень напоминающие *ориентиры* RUP (см. лекции 5).

Приёмочные тесты обычно пишут на обратной стороне карты с соответствующей историей пользователя. Шаблон, используемый в методологии XP, содержит 3 предложения:

- Установка (контекст; инициирующее событие),
- Операция (функция с количественными характеристиками),
- Подтверждение (результаты исполнения функции).

CRC-карты (Класс-Ответсвенность-Кооперация), как и предыдущие 3 артефакта, представляют собой небольшие карточки, в заголовке которых представлено название класса, а ниже – таблица в две колонки. В левой колонке перечислены *ответственности* (т.е. высокоуровневый взгляд на его методы) класса, в правой – классы, состоящие в кооперации с рассматриваемым классом.

Планирование на основе требований на примере XP

Планирование включает следующие работы:

- оценивание,
- планирование версий и итераций.

Оценивание представляет собой определение объёма работ в разрезе историй пользователя. Каждая история оценивается в *пунктах*. Один пункт равен «идеальной» (сорокачасовой) неделе, целиком посвящённой программированию. Если оценка лежит в пределах от 1 до 3 пунктов – то он ставится на карточке истории. Если оценка менее 1 – на карточке ставится 0. Это – так называемый «*песок*». Если оценка превышает 3 пункта – мы имеем дело с «*эпопеей*». В этом случае карточка помечается, как «split» и подлежит процедуре *разделения*. Другая стратегия работы с такой карточкой –попытаться вместить её в оптимальный срок путём исключения декораций. В случае, если история пользователя сложна для экспресс-оценки – необходимо провести исследование или «*гвоздь*» планирования.

Планирование версий и итераций.

Планирование в XP базируется на следующих основных понятиях: производительность, приоритеты, стоимость версии, составление плана версий, составление плана итераций.

Производительность или быстродействие команды базируется на оценках пунктов истории. Однако необходимо учитывать, что пункты представляют идеальные оценки, кроме того существенную роль имеет опыт команды в оценивании (для начинающих команд возможна значительная погрешность).

Ключевую роль в назначении *приоритетов* играет, безусловно, заказчик. Однако и Разработчик имеет право голоса при отборе историй, которые должны попасть в версию (вопросы архитектуры, ключевой функциональности и т.п.).

Стоимость версии определяется, базируясь на производительности, приоритетах и сроках.

План версий даёт Заказчику начальное понимание стоимости проекта. Эта оценка даёт ему возможность отказаться от проекта в начальной его стадии, если сроки и (или) цена являются неприемлемыми.

В случае, если план версий принят – составляется *план итераций*, отражающий шаги (итерации), которые необходимо проделать, чтобы добиться требуемой функциональности продукта.

Анализ требований и управление рисками

Риск в реализации программных проектов – это потенциальная проблема, которая имеет существенную вероятность отрицательно повлиять на успешность проекта, например – на сдачу его в срок, удовлетворение бюджетных ограничений, качество продукта, эффективность работы команды.

Управление риском – комплекс мероприятий по выявлению, оценке, предотвращению и контролю рисков проекта.

Как пишет К.Вигерс [8], «Если что-либо нехорошее уже произошло с вашим проектом, то это – проблема, а не риск… Управление риском означает работу потенциальной опасностью до того, как она перейдет в кризисную фазу». Менеджеры проектов должны выявлять риск и управлять им, начиная с факторов, связанных с требованиями, в сотрудничестве с представителями Заказчика.

Стратегии и работы по управлению риском

Управление риском включает в себя действия, показанные на рис. 15-1 [8].

Рис. 15-1

Работы по *оцениванию* риска (risk assessment) начинаются с *определения* потенциальных опасностей для проекта. В качестве методики выявления может быть рекомендована методика мозгового штурма. Хорошим подспорьем для этого этапа работ является имеющаяся у Разработчика классификация рисков.

Так, все риски принято для делить на прямые (те, на которые Разработчик может так или иначе влиять) и косвенные (независимые от Разработчика) [15].

М. Фаулер [7] предложил разделить все риски на четыре категории:

- риски, связанные с требованиями,
- технологические риски,

- риски, связанные с квалификацией персонала,
- политические риски.

Распространенные факторы риска, связанные с требованиями, включают неверное понимание требований, недостаточное вовлечение пользователей, неточности или изменения в масштабах и целях проекта, постоянно нестабильные требования. Подробный анализ этих видов рисков можно найти в [8], глава 23.

Анализ риска сводится к исследованию и описанию потенциальных последствий конкретных факторов риска для проекта, а также вероятности их проявления.

Определение приоритетов состоит из поиска ответов на два вопроса: насколько вероятно проявление риска в проекте; насколько разрушительны могут быть последствия его проявления.

Обнаруженные риски помещаются в специальный документ – *risk list.*

Существуют три основные стратегии поведения в отношении рисков:
Предотвращение риска, передача риска, принятие риска.

Предотвращение риска (risk avoidance) – это процесс реорганизации проекта таким образом, чтобы риск не мог на него воздействовать. Например – отказаться от вновь появившихся передовых инструментов в пользу испытанных, не включать в план те функции, которые требуют освоения новых технологий.

Передача риска – перераспределение работ проекта таким образом, чтобы кто-то другой (Заказчик, партнёр и т.п.) отвечал за работу с ним.

Принятие риска обязывает Разработчика «заботиться» о нём. Мероприятия по *контролю риска* (risk control) включают планирование, разрешение и мониторинг.

Планирование управления риском подразумевает создание плана действий для каждого отдельного фактора, включая методы смягчения, планы на случай непредвиденных обстоятельств, ответственных лиц и сроки исполнения. Цель действий по смягчению воздействия риска — либо не позволить риску стать проблемой, либо уменьшить его вредное воздействие.

Некоторые риски могут быть *разрешены* в процессе работы над проектом, они удаляются из списка рисков, другие – напротив, обнаружены в ходе выполнения проекта и добавлены в этот документ.

Мониторинг рисков призван осуществлять наблюдение над рисками из списка, отслеживать их продвижение вплоть до разрешения, работать с их приоритетами.

Современные тенденции в развитии АИС и технологий их создания

Индустрия производства АИС претерпела за 2 последних десятилетия качественные изменения. Это связано с такими тенденциями компьютерного мира и мировой экономики, как:

- развитием и появлением новой элементной базы,
- наступлением эпохи персональных компьютеров,
- появлением и развитием интегрированных сред разработки,
- появлением глобальных сетей передачи данных,
- глобализацией бизнеса,
- ростом конкуренции,
- переходом к экономике, ориентированной на потребителя,
- появлением и развитием электронного бизнеса и др.

АИС прошли путь от однопользовательских систем к системам масштаба рабочей группы, малого предприятия, холдинга, транснациональной корпорации.

Среди современных тенденций в развитии технологий создания АИС следует отметить:

- быстрые методы прототипирования,
- ориентацию на варианты использования,
- переход от разработке к настройке.

Покупное или заказное ПО – критерии выбора

В мире IT существует определённая конкуренция между заказными и покупными системами. Основной аргумент сторонников заказных систем – «каждый серьёзный бизнес уникален и требует собственной разработки; универсализм – синоним бедности». Аргументы их противников – «количество типовых решений конечно и невелико. Одни и те же организационные решения работают в различных отраслях промышленности».

Существует значительное количество аргументов в пользу покупных систем, например:

1. ERP-система X разрабатывается вендором Y уже Z лет. За это время он освоил рынки крупнейших стран Запада, позади тысячи внедрений, что говорит о высоком качестве системы.
2. КИС базируется на референтной (эталонной) модели бизнеса. Данная модель апробирована на предприятиях десятков государств и отраслей промышленности и помимо собственно системы вы (покупатель) получите подробные инструкции о том, как правильно вести бизнес и побеждать в конкурентной борьбе.

На порождение этих аргументов работают отделы маркетинга таких гигантов этого сегмента IT-индустрии, как SAP, Oracle, SAGE, Microsoft, SSA Global и др. И эти аргументы действительно и серьёзны и убедительны. Тем не менее, несмотря на мощный прессинг лидеров производства КИС, по данным обзоров, соотношение заказных и покупных систем автоматизации предприятий в мире оценивается, примерно, как 50:50.

По сути, выбор осуществляется даже не из 2, а из 3 вариантов:

1) закупить решение у вендора;
2) заказать эксклюзивное решение у фирмы-производителя прикладного программного обеспечения.
3) изготовить решение самостоятельно.

Основные критерии выбора:

- цена;
- степень уникальности бизнеса компании;
- уровень сервисного обслуживания.

Ценовые вопросы сводятся к анализу затрат на:

- закупку (или разработку),
- мероприятия по внедрению решения,
- владение (включая необходимые доработки).

Степень уникальности бизнеса компании характеризуется степенью отражения особенностей бизнеса компании в решениях, представленных на рынке.

Уровень сервисного обслуживания характеризуется наличием поддержки покупного программного обеспечения по месту размещения предприятия компании, политикой вендора (разработчика) в области поддержки, наличием горячей линии и т.п.

На практике зачастую используется комбинация 1 и 3, либо 2 и 3 вариантов: система закупается, либо разрабатывается на стороне, внедряется, затем её сопровождение и развитие переходит к отделу автоматизации предприятия.

Стратегии выбора решения

Перед специалистом IT-отдела, либо независимым консультантом, которому поручен выбор решения в области автоматизации предприятия, стоит нелёгкая задача. Ведь на рынке представлены сотни решений в области автоматизации и чтобы хотя бы бегло ознакомиться с каждым из них может потребоваться несколько человеко-лет.

На практике, конечно, вряд ли путь подробного изучения всех известных на рынке решений можно рассматривать всерьёз. В простейшем случае рассматриваются аналитические обзоры, подготовленные независимыми экспертами, оцениваются финансовые возможности предприятия внедрения и на третейский суд инвестора предоставляются 2-3 решения.

Чтобы сделать выбор обоснованным, используются стратегии выбора решения. Основные из них:

- анализ требований,
- анализ несоответствия,
- подход на основе «лучших практик».

Анализ требований

Рациональным приёмом, позволяющим снизить затраты на подготовку вариантов решения, а заодно снизить риски, является формирование документа требований (не только ко вновь создаваемому, но и к выбираемому продукту). Тем самым, часть работы можно переложить на представителей маркетинговых отделов вендоров – пусть они объяснят и продемонстрируют, как на практике реализуются Ваши требования.

Практическое применение методов анализа требований, рассмотренных на протяжении лекционного курса, применительно к задаче выбора покупного решения, требует ответа на следующие вопросы:

- рамки проекта;
- широта анализа требований;
- глубина проработки требований;
- требуемые ресурсы.

Рамки проекта. Какие бизнес-процессы, подразделения, отделы предприятия необходимо автоматизировать? Практика внедрения ERP-систем показывает, что даже на самых передовых в информационном смысле предприятиях степень охвата бизнеса информационными технологиями редко превышает порог в 90%. Возможно, крупномасштабный проект автоматизации следует разбить на несколько этапов, начав, допустим, с планки в 30%. Другая стратегия – «всё и сразу» предполагает попытку сразу выйти на максимально возможный охват функций.

Широта анализа требований. Сколько требований вы в состоянии сформулировать в течение разумного промежутка времени? Сколько времени уйдёт у вендоров на анализ документа требований, который вы подготовите? Д. О'Лири в [8] следующий порядок: документ описания требований для предприятия с годовым оборотом в $40 млн., содержит около 1000 позиций (требований).

Глубина проработки требований. Какую выбрать степень детализации требований? Уровень документа «Концепция» вряд ли окажется достаточным для серьёзного рассмотрения. Необходимо отражать как функциональные, та и нефункциональные требования. Можно остановиться на кратких формулировках функций (лаконично, трудно проверяемо). Можно перейти на язык сценариев (появляется возможность доскональной проверки, но это потребует значительных ресурсов). Хорош вариант с комбинацией этих способов описания: критичный функционал описать в форме сценариев, остальное – в виде функций.

Требуемые ресурсы. Ключевой ресурс предприятия – человеческий ресурс. Найдут ли топ-менеджеры предприятия время на скрупулёзную оценку требований к системе? Насколько хорошо смогут сформировать требования руководители линейных подразделений? Насколько удастся задействовать ресурс поставщиков решений? Стоит ли привлекать внешних консультантов?

Какова цена обследования? Как быстро удастся сформировать требования? Какое время следует выделить на получение ответов от вендоров, на анализ этих ответов перед окончательным выбором?

Анализ несоответствия

Анализ несоответствия при выборе КИС базируется на классических методах бизнес-анализа, предпологающих построения моделей «как есть», «как надо» и переходной модели, показывающей путь реформирования предприятия [9].

Нюанс заключается в том, что в данном случае анализ «как есть» применяется не к бизнес-системе в чистом виде, а к её комбинации с имеющейся автоматизированной системой. Он призван высветить слабые стороны и имеющегося решения и связанные с ним проблемы бизнеса.

Анализ «как надо» может производиться по одному из следующих сценариев: «реинжиниринга чистого листа» и «реинжиниринга, запускаемого технологией» [8].

В первом случае используются классические подходы к реинжинирингу предприятий [2,3] в комбинации с методами анализа и формирования требований.

Во втором случае речь идёт о реинжиниринге под конкретное ERP-решение. В этом случае основой для пересмотра действующих бизнес-процессов и уровня их автоматизации являются «лучшие практики», заложенные в соответствующей ERP-системе.

Как и в стратегии анализа требований, стратегия анализа несоответствия оставляет множество подводных камней и вопросов. Допустим, мы составили модель «надо» и рассматриваем спецификации конкурирующих ERP-системы. Как сопоставить эти два документа? Как измерить степень соответствия – по количеству функций? Где гарантии того, что функции, называющиеся одинаково в рассматриваемых документах, *действительно* одинаково работают?

Альтернативой рассмотренным выше стратегиям, позволяющей снизить степень неопределённости принятия решений, в определённой мере является подход на основе лучших практик [8].

Подход на основе лучших практик

Подход основан на отказе от моделей «как есть». Предлагается не зацикливаться на существующих на предприятии бизнес-процессах, а запустить процесс реинжиниринга, основываясь на «лучших практиках» внедряемого ERP-решения. Основные доводы за применение данного подхода [8].

1) Каждый пакет ERP соответствует нуждам организации. Выбранная ERP-система имеет приблизительно тот же набор лучших практик, что и любая другая и все они примерно эквивалентны.

2) Копировать существующие процессы не имеет смысла. Организация должна осуществлять автоматизацию, основываясь на «реинжиниринговом потенциале» ERP-системы.

3) Априорный анализ лучших практик не должен использоваться. Анализ лучших практик должен осуществляться в контексте конкретной части ПО выбранной ERP-системы.

4) Следует не «загибать решение под существующий бизнес», а, напротив, реорганизовать бизнес на основе лучших практик так, чтобы изменения в ERP-системы были минимальны. Это позволит снизить цену внедрения и цену владения ERP-системами.

Процесс выбора решения

Выбор решения для построения на предприятии корпоративной АИС – очень ответственный процесс, связанный с вопросами защиты инвестиций и выживания на рынке. Значительное количество проектов по внедрению ERP-систем заканчивается провалом, особенно это касается российской практики. Более того, на Западе существуют прецеденты, когда предприятия, поставившие «на карту ERP» своё благосостояние и связавшие с ними планы развития, попросту обанкротились, не справившись с задачей «реинжиниринга под ERP».

Поэтому задачу выбора решения следует рассматривать в рамках проектного и процессного подходов, со всеми вытекающими отсюда последствиями.

В формально проработанных процессах внедрения решений недостатка нет. Как правило, они разрабатываются и поставляются вендорами вместе с самими решениями. Вопросы организации выбора решений проработаны в литературе значительно хуже.

Ниже рассмотрен практический процесс, принятый компанией Chesapeake Display & Packaging при выборе ERP-решения – процесс быстрого выбора для быстрого внедрения[30] на базе обзора, представленного в [8].

Процесс организован достаточно просто. Он предусматривает выполнение следующих этапов:

- сформировать команду «выборщиков»;
- организовать демонстрацию КИС поставщиками;
- осуществить предварительный отбор;
- осуществить выбор.

Сформировать команду. Команда должна быть небольшой и тщательно подобранной. Критерии отбора – знание бизнеса и бизнес-процессов; включение людей с различными взглядами, сочетание узкоспециализированных специалистов и специалистов широкого профиля. Ограничение на размеры команды – не более 10 человек.

Организовать демонстрацию КИС поставщиками. Выбор ограниченного количества производителей высококачественных КИС. Предложение выбранным производителям подготовить демонстрацию для команды «выборщиков». Ограничение доступа представителей производителей к членам команды до момента демонстрации. Срок подготовки – 3 недели. Продолжительность демонстрации – 2 дня. Свобода выбора для вендора оборудования и СУБД.

Осуществить предварительный отбор. Поставщикам высказываются следующие требования:

- показать, что КИС сможет работать с вашим бизнесом;
- показать, что вы сможете внедрить его в течение требуемого времени;
- показать своё понимание отрасли.

После проведения демонстрации на основе указанных требований осуществляется выбор тройки лидеров.

Осуществить выбор. Выбор осуществляется на основе голосования, большинством голосов. Голосование производится на основе двух факторов:

- наиболее подходящие функциональные возможности;
- лучший персонал по внедрению.

Каждая из опций ПО оценивается в трёхбалльной шкале (3 – наивысший балл). Оценки осуществляются членами команды по группам компетенции.

Литература к лекции

1. Леффингуелл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. М.: ИД “Вильямс”, 2002.
2. Введение в Rational Unified Process/ Ф. Кратчен – СПб.: Вильямс, 2002. – 240 с.
3. Астелс, Дэвид; Миллер Гранвилл; Новак, Мирослав. Практическое руководство по экстремальному программированию.: Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 320 с.: ил. – Парал. тит. англ.
4. Бек К. Экстремальное программирование. – СПб.: Питер, 2002. – 224 с.
5. Вигерс Карл Разработка требований к программному обеспечению/Пер, с англ. — М.:Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.
6. Л.Новиков. Введение в Rational Unified Process. http://www.interface.ru/rational/interface/151199/rup/main.htm

[30] G.Chemis (1999), Shooting the Rapids of a Rapid Implementation //Shape Your World, Focus ’99 (Denver, CO:J.D.Edwards)

7. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.:Мир, 1999. – 191 с., ил.
8. ERP системы. Современное планирование и управление ресурсами предприятия. Выбор, внедрение, эксплуатация/Дэниел О'Лири;[Пер. с англ. Ю.И.Водопьяновой]. – М.: ООО «Вершина», 2004. – 272 с.
9. Калянов Г. Н. Консалтинг при автоматизации предприятий: Научно-практическое издание. Серия «Информатизация России на пороге XXI века». — М.: СИН-ТЕГ, 1997.
10. Каменова, Громов. Моделирование бизнеса. Методология ARIS. – М.: Весть-МетаТехнология, 2001.