# Этапы разработки проекта: стратегия и анализ

## Введение

Проектирование информационных систем всегда начинается с определения цели проекта. Основная задача любого успешного проекта заключается в том, чтобы на момент запуска системы и в течение всего времени ее эксплуатации можно было обеспечить:

* требуемую функциональность системы и степень адаптации к изменяющимся условиям ее функционирования;
* требуемую пропускную способность системы;
* требуемое время реакции системы на запрос;
* безотказную работу системы в требуемом режиме, иными словами - готовность и доступность системы для обработки запросов пользователей;
* простоту эксплуатации и поддержки системы;
* необходимую безопасность.

Производительность является главным фактором, определяющим эффективность системы. Хорошее проектное решение служит основой высокопроизводительной системы.

Проектирование информационных систем охватывает три основные области:

* проектирование объектов данных, которые будут реализованы в базе данных;
* проектирование программ, экранных форм, отчетов, которые будут обеспечивать выполнение запросов к данным;
* учет конкретной среды или технологии, а именно: топологии сети, конфигурации аппаратных средств, используемой архитектуры (файл-сервер или клиент-сервер), параллельной обработки, распределенной обработки данных и т.п.

В реальных условиях проектирование - это поиск способа, который удовлетворяет требованиям функциональности системы средствами имеющихся технологий с учетом заданных ограничений.

К любому проекту предъявляется ряд абсолютных требований, например максимальное время разработки проекта, максимальные денежные вложения в проект и т.д. Одна из сложностей проектирования состоит в том, что оно не является такой структурированной задачей, как анализ требований к проекту или реализация того или иного проектного решения.

Считается, что сложную систему невозможно описать в принципе. Это, в частности, касается систем управления предприятием. Одним из основных аргументов является изменение условий функционирования системы, например директивное изменение тех или иных потоков информации новым руководством. Еще один аргумент - объемы технического задания, которые для крупного проекта могут составлять сотни страниц, в то время как технический проект может содержать ошибки. Возникает вопрос: а может, лучше вообще не проводить обследования и не делать никакого технического проекта, а писать систему «с чистого листа» в надежде на то, что произойдет некое чудесное совпадение желания заказчика с тем, что написали программисты, а также на то, что все это будет стабильно работать?

Если разобраться, то так ли уж непредсказуемо развитие системы и действительно ли получить информацию о ней невозможно? Вероятно, представление о системе в целом и о предполагаемых (руководством) путях ее развития можно получить посредством семинаров. После этого разбить сложную систему на более простые компоненты, упростить связи между компонентами, предусмотреть независимость компонентов и описать интерфейсы между ними (чтобы изменение одного компонента автоматически не влекло за собой существенного изменения другого компонента), а также возможности расширения системы и «заглушки» для нереализуемых в той или иной версии системы функций. Исходя из подобных элементарных соображений описание того, что предполагается реализовать в информационной системе, уже не кажется столь нереальным. Можно придерживаться классических подходов к разработке информационных систем, один из которых - схема «водопада» (рис. 1) - описан ниже. Кратко будут рассмотрены и некоторые другие подходы к разработке информационных систем, где использование элементов, описанных в схеме «водопада», также допустимо. Какого подхода из описываемых ниже придерживаться (и есть ли смысл придумывать собственный подход) - в какой-то мере дело вкуса и обстоятельств.

Жизненный цикл программного обеспечения представляет собой модель его создания и использования. Модель отражает его различные состояния, начиная с момента возникновения необходимости в данном ПО и заканчивая моментом его полного выхода из употребления у всех пользователей. Известны следующие модели жизненного цикла:

* Каскадная модель. Переход на следующий этап означает полное завершение работ на предыдущем этапе.
* Поэтапная модель с промежуточным контролем. Разработка ПО ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют уменьшить трудоемкость процесса разработки по сравнению с каскадной моделью; время жизни каждого из этапов растягивается на весь период разработки.
* Спиральная модель. Особое внимание уделяется начальным этапам разработки - выработке стратегии, анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования). Каждый виток спирали предполагает создание некой версии продукта или какого-либо его компонента, при этом уточняются характеристики и цели проекта, определяется его качество и планируются работы следующего витка спирали.

Ниже мы рассмотрим некоторые схемы разработки проекта.

## «Водопад» - схема разработки проекта

Очень часто проектирование описывают как отдельный этап разработки проекта между анализом и разработкой. Однако в действительности четкого деления этапов разработки проекта нет - проектирование, как правило, не имеет явно выраженного начала и окончания и часто продолжается на этапах тестирования и реализации. Говоря об этапе тестирования, также следует отметить, что и этап анализа, и этап проектирования содержат элементы работы тестеров, например для получения экспериментального обоснования выбора того или иного решения, а также для оценки критериев качества получаемой системы. На этапе эксплуатации уместен разговор и о сопровождении системы.

Ниже мы рассмотрим каждый из этапов, подробнее остановившись на этапе проектирования.

### Стратегия

Определение стратегии предполагает обследование системы. Основная задача обследования - оценка реального объема проекта, его целей и задач, а также получение определений сущностей и функций на высоком уровне.

На этом этапе привлекаются высококвалифицированные бизнес-аналитики, которые имеют постоянный доступ к руководству фирмы; этап предполагает тесное взаимодействие с основными пользователями системы и бизнес-экспертами. Основная задача взаимодействия - получить как можно более полную информацию о системе (полное и однозначное понимание требований заказчика) и передать данную информацию в формализованном виде системным аналитикам для последующего проведения этапа анализа. Как правило, информация о системе может быть получена в результате бесед или семинаров с руководством, экспертами и пользователями. Таким образом определяются суть данного бизнеса, перспективы его развития и требования к системе.

По завершении основной стадии обследования системы технические специалисты формируют вероятные технические подходы и приблизительно рассчитывают затраты на аппаратное обеспечение, закупаемое программное обеспечение и разработку нового программного обеспечения (что, собственно, и предполагается проектом).

Результатом этапа определения стратегии является документ, где четко сформулировано, что получит заказчик, если согласится финансировать проект; когда он получит готовый продукт (график выполнения работ); сколько это будет стоить (для крупных проектов должен быть составлен график финансирования на разных этапах работ). В документе должны быть отражены не только затраты, но и выгода, например время окупаемости проекта, ожидаемый экономический эффект (если его удается оценить).

В документе обязательно должны быть описаны:

* ограничения, риски, критические факторы, влияющие на успешность проекта, например время реакции системы на запрос является заданным ограничением, а не желательным фактором;
* совокупность условий, при которых предполагается эксплуатировать будущую систему: архитектура системы, аппаратные и программные ресурсы, предоставляемые системе, внешние условия ее функционирования, состав людей и работ, которые обеспечивают бесперебойное функционирование системы;
* сроки завершения отдельных этапов, форма сдачи работ, ресурсы, привлекаемые в процессе разработки проекта, меры по защите информации;
* описание выполняемых системой функций;
* будущие требования к системе в случае ее развития, например возможность работы пользователя с системой с помощью Интернета и т.п.;
* сущности, необходимые для выполнения функций системы;
* интерфейсы и распределение функций между человеком и системой;
* требования к программным и информационным компонентам ПО, требования к СУБД (если проект предполагается реализовывать для нескольких СУБД, то требования к каждой из них, или общие требования к абстрактной СУБД и список рекомендуемых для данного проекта СУБД, которые удовлетворяют заданным условиям);
* что не будет реализовано в рамках проекта.

Выполненная на данном этапе работа позволяет ответить на вопрос, стоит ли продолжать данный проект и какие требования заказчика могут быть удовлетворены при тех или иных условиях. Может оказаться, что проект продолжать не имеет смысла, например из-за того, что те или иные требования не могут быть удовлетворены по каким-то объективным причинам. Если принимается решение о продолжении проекта, то для проведения следующего этапа анализа уже имеются представление об объеме проекта и смета затрат.

Следует отметить, что и на этапе выбора стратегии, и на этапе анализа, и при проектировании независимо от метода, применяемого при разработке проекта, всегда следует классифицировать планируемые функции системы по степени важности. Один из возможных форматов представления такой классификации - MoSCoW - предложен в Clegg, Dai and Richard Barker, Case Method Fast-track: A RAD Approach, Adison-Wesley, 1994.

Эта аббревиатура расшифровывается так: Must have - необходимые функции; Should have - желательные функции; Could have - возможные функции; Won’t have - отсутствующие функции.

Функции первой категории обеспечивают критичные для успешной работы системы возможности.

Реализация функций второй и третьей категорий ограничивается временными и финансовыми рамками: разрабатываем то, что необходимо, а также максимально возможное в порядке приоритета число функций второй и третьей категорий.

Последняя категория функций особенно важна, поскольку необходимо четко представлять границы проекта и набор функций, которые будут отсутствовать в системе.

### Анализ

Этап анализа предполагает подробное исследование бизнес-процессов (функций, определенных на этапе выбора стратегии) и информации, необходимой для их выполнения (сущностей, их атрибутов и связей (отношений)). На этом этапе создается информационная модель, а на следующем за ним этапе проектирования - модель данных.

Вся информация о системе, собранная на этапе определения стратегии, формализуется и уточняется на этапе анализа. Особое внимание следует уделить полноте переданной информации, анализу информации на предмет отсутствия противоречий, а также поиску неиспользуемой вообще или дублирующейся информации. Как правило, заказчик не сразу формирует требования к системе в целом, а формулирует требования к отдельным ее компонентам. Уделите внимание согласованности этих компонентов.

Аналитики собирают и фиксируют информацию в двух взаимосвязанных формах:

* функции - информация о событиях и процессах, которые происходят в бизнесе;
* сущности - информация о вещах, имеющих значение для организации и о которых что-то известно.

Двумя классическими результатами анализа являются:

* иерархия функций, которая разбивает процесс обработки на составные части (что делается и из чего это состоит);
* модель «сущность-связь» (Entry Relationship model, ER-модель), которая описывает сущности, их атрибуты и связи (отношения) между ними.

Эти результаты являются необходимыми, но не достаточными. К достаточным результатам следует отнести диаграммы потоков данных и диаграммы жизненных циклов сущностей. Довольно часто ошибки анализа возникают при попытке показать жизненный цикл сущности на диаграмме ER.

Ниже мы рассмотрим три наиболее часто применяемые методологии структурного анализа:

* диаграммы «сущность-связь» (Entity-Relationship Diagrams, ERD), которые служат для формализации информации о сущностях и их отношениях;
* диаграммы потоков данных (Data Flow Diagrams, DFD), которые служат для формализации представления функций системы;
* диаграммы переходов состояний (State Transition Diagrams, STD), которые отражают поведение системы, зависящее от времени; диаграммы жизненных циклов сущностей относятся именно к этому классу диаграмм.

### ER-диаграммы

ER-диаграммы (рис. 2) используются для разработки данных и представляют собой стандартный способ определения данных и отношений между ними. Таким образом, осуществляется детализация хранилищ данных. ER-диаграмма содержит информацию о сущностях системы и способах их взаимодействия, включает идентификацию объектов, важных для предметной области (сущностей), свойств этих объектов (атрибутов) и их отношений с другими объектами (связей). Во многих случаях информационная модель очень сложна и содержит множество объектов.

Сущность изображается в виде прямоугольника, вверху которого располагается имя сущности (например, TITLES). В прямоугольнике могут быть перечислены атрибуты сущности; атрибуты ER-диаграмм, набранные полужирным шрифтом1, являются ключевыми (так Title Identity - ключевой атрибут сущности TITLES, остальные атрибуты ключевыми не являются).

Отношение изображается линией между двумя сущностями (синие линии на рисунке).

Одиночная линия справа (рис. 3) означает «один», «птичья лапка» слева - «многие», а отношение читается вдоль линии, например «один ко многим». Вертикальная черта означает «обязательно», кружок - «не обязательно», например для каждого издания в TITLE обязательно должен быть указан издатель в PUBLISHERS, а один издатель в PUBLISHERS может выпускать несколько наименований изданий в TITLES. Следует отметить, что связи всегда комментируются (надпись на линии, изображающей связь).

Рис. 3. Элемент ER-диаграммы

Приведем также пример (рис. 4) изображения рефлексивного отношения «сотрудник», где один сотрудник может руководить несколькими подчиненными и так далее вниз по иерархии должностей.

Следует обратить внимание на то, что такое отношение всегда является необязательным, в противном случае это будет бесконечная иерархия.

Атрибуты сущностей могут быть ключевыми - они выделяются полужирным шрифтом; обязательными - перед ними ставится знак «\*», то есть их значение всегда известно, необязательными (optional) - перед ними ставится О, то есть значения этого атрибута в какие-то моменты могут отсутствовать или быть неопределенными.

### Дуги

Если сущность имеет набор взаимоисключающих отношений с другими сущностями, то говорят, что такие отношения находятся в дуге. Например, банковский счет может быть оформлен или для юридического лица, или для физического лица. Фрагмент ER-диаграммы для такого типа отношений приведен на рис. 5.

В этом случае атрибут ВЛАДЕЛЕЦ сущности СЧЕТ имеет особое значение для данной сущности - сущность делится на типы по категориям: «для физического лица» и «для юридического лица». Полученные в результате сущности называют подтипами, а исходная сущность становится супертипом. Чтобы понять, нужен супертип или нет, надо установить, сколько одинаковых свойств имеют различные подтипы. Следует отметить, что злоупотребление подтипами и супертипами является довольно распространенной ошибкой. Изображают их так, как показано на рис. 6.

### Нормализация

Чтобы не допустить аномалий при обработке данных, используют нормализацию. Принципы нормализации для объектов информационной модели в точности такие же, как и для моделей данных.

Допустимые типы связей. При ближайшем рассмотрении связи типа «один к одному» (рис. 7) почти всегда оказывается, что A и B представляют собой в действительности разные подмножества одного и того же предмета или разные точки зрения на него, просто имеющие отличные имена и по-разному описанные связи и атрибуты.

Связи «многие к одному» представлены на рис. 8.

I - достаточно сильная конструкция, предполагающая, что вхождение сущности B не может быть создано без одновременного создания по меньшей мере одного связанного с ним вхождения сущности A.

II - это наиболее часто встречающаяся форма связи. Она предполагает, что каждое и любое вхождение сущности A может существовать только в контексте одного (и только одного) вхождения сущности B. В свою очередь, вхождения B могут существовать как в связи с вхождениями A, так и без нее.

III - применяется редко. Как A, так и B могут существовать без связи между ними.

Связи «многие ко многим» представлены на рис. 9.

I - такая конструкция часто имеет место в начале этапа анализа и означает связь - либо понятую не до конца и требующую дополнительного разрешения, либо отражающую простое коллективное отношение - двунаправленный список.

II - применяется редко. Такие связи всегда подлежат дальнейшей детализации.

Рассмотрим теперь рекурсивные связи (рис. 10).

I - редко, но имеет место. Отражает связи альтернативного типа.

II - достаточно часто применяется для описания иерархий с любым числом уровней.

III - имеет место на ранних этапах. Часто отражает структуру «перечня материалов» (взаимная вложенность компонентов). Пример: каждый КОМПОНЕНТ может состоять из одного и более (других) КОМПОНЕНТОВ и каждый КОМПОНЕНТ может использоваться в одном и более (других) КОМПОНЕНТОВ.

Недопустимые типы связей. К недопустимым типам связей относятся следующие: обязательная связь «многие ко многим» (рис. 11) и ряд рекурсивных связей (рис. 12).

Обязательная связь «многие ко многим» в принципе невозможна. Такая связь означала бы, что ни одно из вхождений A не может существовать без B, и наоборот. На деле каждая подобная конструкция всегда оказывается ошибочной.

### Диаграммы потоков данных

Логическая DFD (рис. 13) показывает внешние по отношению к системе источники и стоки (адресаты) данных, идентифицирует логические функции (процессы) и группы элементов данных, связывающие одну функцию с другой (потоки), а также идентифицирует хранилища (накопители) данных, к которым осуществляется доступ. Структуры потоков данных и определения их компонентов хранятся и анализируются в словаре данных. Каждая логическая функция (процесс) может быть детализирована с помощью DFD нижнего уровня; когда дальнейшая детализация перестает быть полезной, переходят к выражению логики функции при помощи спецификации процесса (мини-спецификации). Содержимое каждого хранилища также сохраняют в словаре данных, модель данных хранилища раскрывается с помощью ER-диаграмм.

В частности, в DFD не показываются процессы, которые управляют собственно потоком данных и не приводятся различия между допустимыми и недопустимыми путями. DFD содержат множество полезной информации, а кроме того:

* позволяют представить систему с точки зрения данных;
* иллюстрируют внешние механизмы подачи данных, которые потребуют наличия специальных интерфейсов;
* позволяют представить как автоматизированные, так и ручные процессы системы;
* выполняют ориентированное на данные секционирование всей системы.

Потоки данных используются для моделирования передачи информации (или даже физических компонентов) из одной части системы в другую. Потоки на диаграммах изображаются именованными стрелками, стрелки указывают направление движения информации. Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться в ее источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним двунаправленным.

Процесс преобразует входной поток данных в выходной в соответствии с действием, задаваемым именем процесса. Каждый процесс должен иметь уникальный номер для ссылок на него внутри диаграммы. Этот номер может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Хранилище данных (data storage) позволяет на ряде участков определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информацию, которую оно содержит, можно использовать в любое время после ее определения, при этом данные могут выбираться в произвольном порядке. Имя хранилища должно идентифицировать его содержимое. В случае когда поток данных входит (выходит) в (из) хранилище и его структура соответствует структуре хранилища, он должен иметь то же самое имя, которое нет необходимости отражать на диаграмме.

Внешняя сущность (терминатор) представляет сущность вне контекста системы, являющуюся источником или приемником системных данных. Ее имя должно содержать существительное, например «Клиент». Предполагается, что объекты, представленные такими узлами, не должны участвовать ни в какой обработке.

### Диаграммы изменения состояний STD

Жизненный цикл сущности относится к классу STD-диаграмм (рис. 14). Эта диаграмма отражает изменение состояния объекта с течением времени. Например, рассмотрим состояние товара на складе: товар может быть заказан у поставщика, поступить на склад, храниться на складе, проходить контроль качества, может быть продан, забракован, возвращен поставщику. Стрелки на диаграмме показывают допустимые изменения состояний.

Существует несколько различных вариантов изображения подобных диаграмм, на рисунке приведен лишь один из них.

## Некоторые принципы проверки качества и полноты информационной модели (источник - Richard Barker, Case Method: Entity Relationship Modelling, Addison-Wesley, 1990)

Если вы хотите создать качественную модель, то придется прибегать к помощи аналитиков, хорошо владеющих CASE-технологией. Однако это не означает, что построением и контролем информационной модели должны заниматься только аналитики. Помощь коллег также может оказаться весьма полезной. Привлекайте их к проверке поставленной цели и к детальному изучению построенной модели как с точки зрения логики, так и с точки зрения учета аспектов предметной области. Большинство людей легче находят недостатки в чужой работе.

Регулярно представляйте вашу информационную модель или ее отдельные фрагменты, относительно которых у вас возникают сомнения, на одобрение пользователей. Особое внимание уделяйте исключениям из правил и ограничениям.

### Качество сущностей

Основной гарантией качества сущности является ответ на вопрос, действительно ли объект является сущностью, то есть важным объектом или явлением, информация о котором должна храниться в базе данных.

Список проверочных вопросов для сущности:

* Отражает ли имя сущности суть данного объекта?
* Нет ли пересечения с другими сущностями?
* Имеются ли хотя бы два атрибута?
* Всего атрибутов не более восьми?
* Есть ли синонимы/омонимы данной сущности?
* Сущность определена полностью?
* Есть ли уникальный идентификатор?
* Имеется ли хотя бы одна связь?
* Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, архивированию и использованию значения сущности?
* Ведется ли история изменений?
* Имеет ли место соответствие принципам нормализации данных?
* Нет ли такой же сущности в другой прикладной системе, возможно, под другим именем?
* Не имеет ли сущность слишком общий смысл?
* Достаточен ли уровень обобщения, воплощенный в ней?

Список проверочных вопросов для подтипа:

* Отсутствуют ли пересечения с другими подтипами?
* Имеет ли подтип какие-нибудь атрибуты и/или связи?
* Имеют ли они все свои собственные уникальные идентификаторы или наследуют один на всех от супертипа?
* Имеется ли исчерпывающий набор подтипов?
* Не является ли подтип примером вхождения сущности?
* Знаете ли вы какие-нибудь атрибуты, связи и условия, отличающие данный подтип от других?

### Качество атрибутов

Следует выяснить, а действительно ли это атрибуты, то есть описывают ли они тем или иным образом данную сущность.

Список проверочных вопросов для атрибута:

* Является ли наименование атрибута существительным единственного числа, отражающим суть обозначаемого атрибутом свойства?
* Не включает ли в себя наименование атрибута имя сущности (этого быть не должно)?
* Имеет ли атрибут только одно значение в каждый момент времени?
* Отсутствуют ли повторяющиеся значения (или группы)?
* Описаны ли формат, длина, допустимые значения, алгоритм получения и т.п.?
* Не может ли этот атрибут быть пропущенной сущностью, которая пригодилась бы для другой прикладной системы (уже существующей или предполагаемой)?
* Не может ли он быть пропущенной связью?
* Нет ли где-нибудь ссылки на атрибут как на «особенность проекта», которая при переходе на прикладной уровень должна исчезнуть?
* Есть ли необходимость в истории изменений?
* Зависит ли его значение только от данной сущности?
* Если значение атрибута является обязательным, всегда ли оно известно?
* Есть ли необходимость в создании домена для этого и ему подобных атрибутов?
* Зависит ли его значение только от какой-то части уникального идентификатора?
* Зависит ли его значение от значений некоторых атрибутов, не включенных в уникальный идентификатор?

### Качество связи

Нужно выяснить, отражают ли связи действительно важные отношения, наблюдаемые между сущностями.

Список проверочных вопросов для связи:

* Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи и вписывается ли в принятый синтаксис?
* Участвуют ли в ней только две стороны?

Не является ли связь переносимой?

* Заданы ли степень связи и обязательность для каждой стороны?
* Допустима ли конструкция связи?

Не относится ли конструкция связи к редко используемым?

* Не является ли она избыточной?
* Не изменяется ли она с течением времени?
* Если связь обязательная, всегда ли она отражает отношение к сущности, представляющей противоположную сторону?

Для исключающей связи:

* Все ли концы связей, покрываемые исключающей дугой, имеют один и тот же тип обязательности?
* Все ли из них относятся к одной и той же сущности?
* Обычно дуги пересекают разветвляющиеся концы - что вы можете сказать о данном случае?
* Связь может покрываться только одной дугой. Так ли это?
* Все ли концы связей, покрываемые дугой, входят в уникальный идентификатор?

## Функции системы

Часто аналитикам приходится описывать достаточно сложные бизнес-процессы. В этом случае прибегают к функциональной декомпозиции, которая показывает разбиение одного процесса на ряд более мелких функций до тех пор, пока каждую из них уже нельзя будет разбить без ущерба для смысла. Конечный продукт декомпозиции представляет собой иерархию функций, на самом нижнем уровне которой находятся атомарные с точки зрения смысловой нагрузки функции. Приведем простой пример (рис. 15) такой декомпозиции. Рассмотрим простейшую задачу выписки счета клиенту при отпуске товара со склада при условии, что набор товаров, которые хочет приобрести клиент, уже известен (не будем рассматривать в данном примере задачу выбора товаров).

Очевидно, что операция выбора и расчета скидок может быть также разбита на более мелкие операции, например на расчет скидок за приверженность (клиент покупает товары в течение долгого времени) и на расчет скидок за количество покупаемого товара. Атомарные функции описываются подробно, например с помощью DFD и STD. Очевидно, что такое описание функций не исключает и дополнительное словесное описание (например, комментарии).

Следует отметить, что на этапе анализа следует уделить внимание функциям анализа и обработки возможных ошибок и отклонений от предполагаемого эталона работы системы. Следует выделить наиболее критичные для работы системы процессы и обеспечить для них особенно строгий анализ ошибок. Обработка ошибок СУБД (коды возврата), как правило, представляет собой обособленный набор функций или одну-единственную функцию.

## Уточнение стратегии

На этапе анализа происходит уточнение выбранных для конечной реализации аппаратных и программных средств. Для этого могут привлекаться группы тестирования, технические специалисты. При проектировании информационной системы важно учесть и дальнейшее развитие системы, например рост объемов обрабатываемых данных, увеличение интенсивности потока запросов, изменение требований надежности информационной системы.

На этапе анализа определяются наборы моделей задач для получения сравнительных характеристик тех или иных СУБД, которые рассматривались на этапе определения стратегии для реализации информационной системы. На этапе определения стратегии может быть осуществлен выбор одной СУБД. Данных о системе на этапе анализа уже намного больше, и они более подробны. Полученные данные, а также характеристики, переданные группами тестирования, могут показать, что выбор СУБД на этапе определения стратегии был неверным и что выбранная СУБД не может удовлетворять тем или иным требованиям информационной системы. Такие же данные могут быть получены относительно выбора аппаратной платформы и операционной системы. Получение подобных результатов инициирует изменение данных, полученных на этапе определения стратегии, например пересчитывается смета затрат на проект.

Выбор средств разработки также уточняется на этапе анализа. В силу того что этап анализа дает более полное представление об информационной системе, чем оно было на этапе определения стратегии, план работ может быть скорректирован. Если выбранное на предыдущем этапе средство разработки не позволяет выполнить ту или иную часть работ в заданный срок, то принимается решение об изменении сроков (как правило, это увеличение срока разработки) или о смене средства разработки. Осуществляя выбор тех или иных средств, следует учитывать наличие высококвалифицированного персонала, который владеет выбранными средствами разработки, а также наличие администраторов выбранной СУБД. Эти рекомендации также будут уточнять данные этапа выбора стратегии (совокупность условий, при которых предполагается эксплуатировать будущую систему).

Уточняются также ограничения, риски, критические факторы. Если какие-либо требования не могут быть удовлетворены в информационной системе, реализованной с использованием СУБД и программных средств, выбранных на этапе определения стратегии, то это также инициирует уточнение и изменение получаемых данных (в конечном итоге сметы затрат и планов работ, а возможно, и изменение требований заказчика к системе, например их ослабление). Более подробно описываются те возможности, которые не будут реализованы в системе.