ВЫСШАЯ ШКОЛА ЭКОНОМИКИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

С.В. Зыков

ОСНОВЫ ПРОЕКТИРОВАНИЯ КОРПОРАТИВНЫХ СИСТЕМ

Издательский дом Высшей школы экономики
Москва 2012

УДК 004.9
ББК 32.97
З-96

Рецензент — кандидат технических наук,
заведующий кафедрой информационных систем
и технологий Нижегородского филиала НИУ ВШЭ
Э.А. Бабкин

З-96 **Зыков, С. В.** Основы проектирования корпоративных систем [Текст] / С. В. Зыков ; Нац. исслед. ун-т «Высшая школа экономики». — М. : Изд. дом Высшей школы экономики, 2012. — 431, [1] с. — 600 экз. — ISBN 978-5-7598-0862-6 (в обл.).

В монографии рассматриваются важнейшие аспекты разработки прикладных программных систем для корпораций — крупных распределенных индустриальных структур, объединенных общими бизнес-целями. Особенностью подхода является исследование всего комплекса архитектурных уровней, необходимых для построения таких систем, — от моделей жизненного цикла и методологий их реализации до технологических платформ и инструментальных средств. Приведен ряд примеров, иллюстрирующих особенности применения современных технологий (в первую очередь, разработанных корпорацией Microsoft) для реализации и внедрения крупномасштабных программных систем в различных отраслях народного хозяйства.

Для студентов, аспирантов и исследователей, а также специалистов-практиков, область интересов которых связана с разработкой крупномасштабных программных систем.

УДК 004.9
ББК 32.97

ISBN 978-5-7598-0862-6

ОГЛАВЛЕНИЕ

Раздел III
ПРИМЕРЫ ОТРАСЛЕВЫХ ВНЕДРЕНИЙ КОРПОРАТИВНЫХ СИСТЕМ

ПРЕДИСЛОВИЕ

Монография преследует несколько взаимосвязанных целей. Прежде всего, это систематизация знаний из областей, связанных с проектированием крупномасштабных, распределенных программных комплексов, состоящих из разнородных компонентов. На основе проведенной систематизации открывается возможность классификации программных систем с последующим методологическим обобщением для жизненного цикла исследуемого класса программных комплексов, которые как по характеру, так и по масштабу своего применения вполне соответствуют термину «корпоративные».

Современные корпоративные системы — это петабайты (тысячи терабайт) данных, как минимум, удвоение объема информации каждые пять лет, необходимость адаптации к требованиям бизнеса практически «на лету», архитектурная и структурная разнородность компонентов и, конечно, глобальная распределенность. В подобных условиях обобщения до уровня технологических схем и инструментальных средств, как правило, не приводят к адекватной интеграции гетерогенных приложений в унифицированные, крупномасштабные программные системы. Поэтому для управления жизненным циклом корпоративных программных комплексов необходимо прежде всего получить адекватное обобщение на уровне математических моделей с последующей конкретизацией представлений на уровне системной архитектуры, информационных технологий, а также конкретных инструментальных и программных средств и, наконец, практических внедрений.

Предлагаемое в настоящей работе методологическое обобщение включает весь спектр возможных (и, как оказывается, необходимых) уровней представления данных в корпоративных программных комплексах в ходе их разработки. К таковым следует отнести прежде всего уровни:

1) математического моделирования концептуального представления;

2) анализа и проектирования архитектурной схемы;

3) реализации, интеграции и адаптации/расширения программных систем.

Естественно, все перечисленные уровни необходимы, взаимозависимы и взаимосвязаны: математические модели адекватно

поддержаны промышленными технологиями, инструментально-программными средствами и приводят к отраслевым внедрениям гетерогенных корпоративных программных комплексов с практически приемлемыми эксплуатационными характеристиками.

Для построения генерализации корпоративных программных комплексов на уровне моделей — как для представления данных, так и для манипулирования ими — потребовался творческий синтез основных положений целого ряда математических теорий, связанных с объектными моделями. В основе рассматриваемых моделей лежит так называемый аппликативный подход к вычислениям, при котором в фокусе внимания находится операция приложения функции к аргументу. Дальнейшее развитие созданной методологии разработки программных систем корпоративного типа — поддержка разработанных моделей программными технологиями, системными архитектурами и инструментальными средствами.

Структура книги во многом является отражением именно такого, методологически последовательного и концептуально наполненного подхода к организации жизненного цикла прикладных корпоративных систем — от моделей до примеров внедрения. Именно таким образом — путем последовательного уточнения — и происходит разработка прикладных систем, при этом окончательный вид конкретной детализации каждого элемента обобщенной методологии зависит от характера и масштаба приложений.

Автор выражает искреннюю благодарность ведущим экспертам университета Carnegie Mellon (G. Taran, N. Bier и др.), а также профессорам С.М. Авдошину (НИУ ВШЭ) и В.Э. Вольфенгагену (МИФИ, МФТИ) за методическую помощь при подготовке монографии.

Отдельно следует отметить целый ряд студентов НИУ ВШЭ и МФТИ (в первую очередь К. Лузгину, Е. Первушину, А. Ядройцева, а также многих других), ассистировавших при предварительном редактировании теста и активно способствовавших выходу в свет настоящей публикации.

Автор полагает, что книга окажется полезной системным аналитикам и архитекторам, разработчикам крупномасштабных приложений, а также студентам, аспирантам и другим категориям исследователей корпоративных программных систем и комплексов.

Раздел I
МОДЕЛИ, МЕТОДОЛОГИИ И АРХИТЕКТУРЫ РАЗРАБОТКИ КОРПОРАТИВНЫХ СИСТЕМ

Глава 1
ВВЕДЕНИЕ В РАЗРАБОТКУ КОРПОРАТИВНЫХ СИСТЕМ

В представленной монографии описана разработка корпоративных информационных систем, даны понятия информационных систем вообще и корпоративных в частности, а также то, для чего, собственно, нам необходимы эти знания и как их применять непосредственно в разработке.

Основные понятия, которые будут представлены далее, — это информационная система и корпорация, т.е. большое, территориально распределенное предприятие с общими задачами, для которого развитие информационных систем, их жизненного цикла происходит несколько иначе, чем информационных систем вообще. В монографии будет рассказано о жизненном цикле информационных систем, относящихся к классу корпоративных приложений, а также основных методологических подходах к их проектированию, реализации и разработке.

Предметом исследования выступают корпоративные информационные системы. Здесь значимо каждое слово. И корпоративная специфика — специфика крупных распределенных предприятий, и понятие системы, т.е. достаточно сложной совокупности различных компонентов, и, конечно, собственно информационной системы как вполне определенного подвида. Назначение данной книги состоит в том, чтобы: дать базовые, фундаментальные представления о предмете, продемонстрировать применение моделей к разработке корпоративных систем, правда, не совсем в математическом понимании этого слова; ввести основные понятия, обозначить их взаимосвязи и применение к корпоративной специфике, описать современную теорию и практику разработки корпоративных приложений.

Будут описаны приложения, т.е. специфичные информационные системы, которые играют существенную прикладную роль. Будет говориться как о практическом аспекте разработки этих приложений, массе аспектов, которые связаны с жизненным циклом системы, во взаимосвязи этапов развития и проектирования и реализации такого рода информационных систем, так и о теоретическом минимуме, который необходим для правильного понимания того, откуда эти системы возникли, как они развиваются, работают и почему нам необходимо планомерно их расширять, создавать и развивать.

Цель курса — формирование целостной концепции проектирования и реализации, иными словами, разработки, корпоративных приложений в современных условиях. Будет применяться подход Карнеги-Мелонского университета к дисциплине «Software engineering» (программная инженерия), в том смысле, что некоторые его математические модели будут далее упомянуты. В целом необходимо сосредоточиться на теории, которая адекватно поддерживает практику, и именно в том необходимом объеме, который поможет правильно составить систему понятий. А также придерживаться этой системы в соответствии с той многочисленной литературой, которая имеется по предмету, для того чтобы иметь возможноть грамотно и с хорошим теоретическим обоснованием говорить о таком непростом и многообразном предмете, как корпоративные информационные системы.

Кроме того, речь пойдет о разработке хорошего стиля проектирования систем. Под хорошим стилем нужно понимать многофакторную оптимизацию, т.е. достаточно сложный процесс — совокупность процессов, которые сопровождают развитие информационных систем. В любом проекте по разработке корпоративных систем имеет место треугольник компромиссов: функциональность, стоимость, временны́е затраты. То есть существует ряд ограничений, которые необходимо учитывать, и для создания хорошего продукта следует уметь находить компромисс.

Кроме того, когда речь пойдет о жизненном цикле программных систем, этапах их развития, проектирования, реализации, внедрения и сопровождения, будут описаны практические примеры их внедрения, а также даны необходимые знания, которые требуются для получения основных навыков, характеризующих такого важного специалиста в разработке корпоративных информационных систем, как

системный архитектор — человек, который отвечает за проектирование информационных систем с технологической точки зрения.

Естественно, предмет, имеющий технический характер и нацеленный на разработку приложений, прикладных систем, предназначен преимущественно для студентов и магистрантов, которые специализируются на разработке приложений.

Необходимая основа — некоторое понимание моделей, на которых базируется объектный подход к проектированию и реализации программных систем. С математической точки зрения моделью объекта может являться функция. Часто речь будет идти о λ-исчислении и других моделях об исчислении функций, которые представляют собой фундамент моделирования, в том числе информационных систем, и вообще фундамент для объектных моделей. Что касается основ проектирования и программирования — желательны знания в области объектно-ориентированного подхода к проектированию и разработке приложений и основ технологий Microsoft, в частности Microsoft.NET.

В конце книги будут даны некоторые практические решения: на базе группы команий «Итера» и на основе технологий Microsoft, в том числе программного обеспечения Microsoft Dynamics. Будет использована открытая информация, которая имеется на сайтах Microsoft, связанных с Dynamics, а также другие открытые источники информации.

Во введении говорится о корпорации как о большой распределенной группе компаний с общими задачами, о программных системах, приложениях как конкретизации этого понятия для решения тех или иных прикладных задач, о жизненном цикле этих систем — от анализа и проектирования до реализации, внедрения, сопровождения и вывода из эксплуатации. Также речь идет о документации, которая сопровождает этот жизненный цикл и без которой, собственно говоря, программный продукт, передаваемый заказчику, таковым не является. Рассказано, что такое методология, как можно ее понимать в узком и широком смысле. Обсуждены основные этапы жизненного цикла программных систем, в том числе соответствующие жизненному циклу корпоративных систем, и, кроме того, представлены основные методологии разработки прикладных программных систем.

Описание корпоративных информационных системах следует начать с понятия корпораций. Под корпорацией понимается

крупная (от 1000 сотрудников), территориально распределенная (часто по всему земному шару) глобальная, трансконтинентальная производственная структура, т.е. бизнес-структура, которая нацелена на производство продукции. Достаточно большие корпорации, например нефтегазовые, — Газпром, известная крупная и распределенная структура реального сектора экономики, TNK-BP, где объединены ресурсы как отечественных, так и зарубежных производителей нефти и газа, и нефтегазовая группа компаний «Итера», объединяющая порядка 150 компаний в 24 странах мира с общей численностью персонала 10 тыс. человек. Несмотря на распределенность структуры, у корпорации, как правило, есть единый центр управления и общие бизнес-цели и задачи.

Следует отметить, что корпорацию можно понимать и шире. Можно относить к корпорациям и не вполне производственные, но тем не менее распределенные и объединенные общими бизнес-целями структуры. С точки зрения корпоративных информационных систем это вполне допустимый угол зрения. Так, в качестве примера корпоративной структуры можно рассматривать Минпромэнерго, научно-исследовательские учреждения, такие как ИПУ РАН, который включают не менее 10 филиалов по России, университеты, такие как МИФИ, НИУ ВШЭ. ВШЭ имеет около 20 различных площадок только в Москве, МИФИ является также достаточно распределенной структурой с точки зрения как студенческого городка, так и распределенности по территории России. Все эти и другие аналогичные структуры вполне можно отнести к корпорациям.

Теперь остановимся на программных системах. Они представляют собой совокупность программ под общим управлением, т.е. примерно так же, как и корпорация как группа компаний — это совокупность взаимодействующих программ. При этом программная система предназначена для решения либо некой замкнутой задачи, либо целого ряда взаимосвязанных задач. И если мы говорим о корпоративной системе, там таких задач достаточно много: это учет, планирование и управление различного рода ресурсами, прежде всего людскими, финансовыми и специфическими производственными ресурами, основными средствами, процессом производства и целым рядом других процессов.

Приложения — это несколько более узкая сущность, нежели просто программа. Они предназначены для решения функциональ-

ных задач по обработке информации в рамках той или иной предметной области. Приложение так или иначе связано с предметной областью. Поэтому если говорить о корпоративных приложениях, корпоративных системах, то речь пойдет о решении функциональных бизнес-задач корпорации с помощью специализированных в каждом случае прикладных программ.

Некоторые отрасли и задачи, такие как планирование, управление, организация, учет, уже были перечислены. Конечно, в масштабе корпорации все эти контуры управления имеет смысл рассматривать не изолированно, а интегрированно. Таким образом, получаются интегрированные или согласованные программные системы, и открывается возможность консолидации данных и построения консолидированных отчетов, например, по персоналу в рамках корпорации в целом, отдельных ее компаний, более мелких подразделений, или такой же детализации, скажем, связанной с финансовой отчетностью и отчетностью по другим производственным ресурсам.

В любой информационной системе, и корпоративной в том числе, каждое приложение, которое функционирует в рамках этой системы, проходит определенные этапы своего становления и развития. Как правило, последовательность этих этапов остается неизменной. Хотя в зависимости от методологий проектирования и разработки применяемых корпоративных систем эта последовательность может претерпевать незначительные изменения, существует ряд этапов и фазы, которые определяют жизненный цикл программных систем. Первым этапом является постановка задачи, т.е. определение характера и масштаба того программного решения, которое будет разработано. Следующий этап связан с анализом требований, которые выдвигает заказчик к программной системе, с анализом того, насколько эти требования адекватны, корректны, полны, непротиворечивы, насколько они соответствуют характеру и масштабу проблемы и полно и точно описывают ту предметную область и задачу, которую должно решать программное обеспечение. Исходя из анализа требований строятся проектные спецификации — это следующий этап жизненного цикла программных систем. На основе проектных спецификаций в форме диаграмм, сценариев использования, т.е. определенных достаточно формально изложенных текстовых данных и технического задания в том числе, происходит проектирование программной

системы. Это следующий этап жизненного цикла — построение в некотором абстрактном виде, прежде всего в форме диаграмм, основных функций, которые эта система будет реализовывать. По сути, речь идет о поэтапной детализации от абстрактного представления к конкретному в ходе развития системы вдоль ее жизненного цикла.

После проектирования, т.е. после того, как все основные объекты, которые будут реализованы в рамках информационной системы, получили свои очертания и становится известно, какими характеристиками они будут обладать, как будут взаимодействовать друг с другом, начинается этап реализации. Это — программирование, кодирование отдельных частей программной системы, которая декомпозирована на них в ходе этапа проектирования. Реализацию сопровождает тестирование программного продукта, которое призвано ответить на вопрос, насколько это программное обеспечение корректно, нет ли в нем внутренних ошибок. Еще один важный вопрос, на который должно ответить тестирование, — в какой мере разработанное программное обеспечение соответствует тем проектным спецификациям и требованиям, которые были к нему сформулированы. Если тестирование не выявляет существенного количества критического уровня ошибок, то можно переходить к фазе передачи заказчику, когда на основе специально созданной серии приемочных тестов осуществляют приемку и передачу в эксплуатацию созданной программной системы или компонента корпоративной информационной системы.

После передачи заказчику наступает фаза сопровождения, которая собственно и является основной, наиболее важной, значимой и затратной частью жизненного цикла (ЖЦ) программной системы. Нужно понимать, что ЖЦ — это процесс прежде всего непрерывный, т.е. переход от стадии к стадии обязан происходить и происходит достаточно плавно, каждая предыдущая стадия является основой для последующей, в том числе и документация, которая выступает достаточно важной составляющей по программному продукту, во многих случаях является «сырьем» для начала и успешного завершения следующей стадии. Жизненный цикл — процесс замкнутый, потому как достаточно сложно переходить с одной стадии к другой, миновав промежуточную. Если говорить о корпоративных системах, то, как правило, это не вполне корректный переход. Кроме того, еще одна важная особенность жизнен-

ного цикла — его итеративность, т.е. то, что жизненный цикл происходит итерационно.

В ряде моделей жизненного цикла приходится иметь дело с последовательным приближением решения к цели. На определенном этапе получается не полномасштабный с точки зрения функциональности программный продукт, но продукт, который в полной мере уже можно назвать продуктом в том смысле, что он проходит все перечисленные стадии и после того, как передан на сопровождение в виде первичном, ограниченном по функциональности, продолжает развиваться и эволюционировать — начиная с коррекции анализа требований и проектных спецификаций. То есть происходит повторное проектирование и, по сути, повтор всего ЖЦ. Жизненный цикл нужно понимать как процесс или смену фаз, которая происходит во времени последовательно. Фазы, естественно, взаимосвязаны. Важная взаимосвязь между фазами определяется проектной документацией и документацией по программному продукту.

Если говорить о процессе разработки информационной системы, в том числе и корпоративной, то, по сути, в широком смысле речь идет о полном жизненном цикле. В разработке корпоративных информационных систем предполагается, что речь идет о жизненном цикле от начала до завершения в широком смысле. Однако если говорить более узко, то можно рассматривать лишь ту часть жизненного цикла, которая связана с кодированием и программированием, но в случае корпоративных систем это является слишком ограниченным подходом. Какие стадии связаны непосредственно с кодированием или программированием? Это проектирование системы, когда речь уже идет о построении скелета реализации, некоего наброска основы разрабатываемой программной системы, т.е., по сути, речь идет уже о работе с CASE-средствами, со средствами автоматизированного проектирования программного обеспечения, которые обладают возможностями кодогенерации для некоторых стандартных компонентов, составляющих основу программной системы. Затем происходит реализация системы — кодирование. После этого реализация ведется вместе с тестированием, и после реализации фрагментов программной системы происходит их интеграция, сборка, которая также связана с тестированием, и передача заказчику. Может быть также стадия сопровождения.

Важный вывод, который можно сделать из сказанного, состоит в том, что для экономичной разработки корпоративных приложений в контексте корпоративных информационных систем, которые являются очень крупными и сложными, имеют большое количество взаимодействующих компонентов, необходимо представлять всю схему жизненного цикла, начиная от постановки задачи, анализа и спецификации требований и заканчивая передачей заказчику, сопровождением и выводом из эксплуатации. Только при полном понимании того, как организованы эти стадии, каждая из них, как они взаимодействуют, в рамках каких моделей они существуют, можно сформировать адекватное представление о функционировании этих систем, их разработке и понять, каким образом эту разработку сделать более экономичной. Этот аспект является крайне важным в проекте и вдвойне актуальным, учитывая текущую кризисную ситуацию, когда бюджеты на разработку систем урезаются. Можно найти возможность сэкономить, чтобы система стала ненамного хуже по таким показателям, как надежность, масштабируемость, вычислительная эффективность, эргономичность, функциональность, безопасность и сопровождаемость, документируемость, однако ее разработка заняла бы более короткое время или позволила высвободить людские ресурсы.

Еще одно важное определение — это методолгия разработки информационных систем — корпоративных и преимущественно ИС. Под методологией будем понимать подход, подразумевающий совокупность методов или практических приемов, нацеленный на завершение отдельно взятой фазы, или стадии, или ряда стадий, которые могут быть взаимосвязаны, жизненного цикла программного обеспечения.

Фазы ЖЦ ПО только что были перечислены — это анализ и спецификация требований, первичное детальное проектирование, реализация вместе с тестированием, интеграция или сборка, когда появляется сначала частичный, затем полный программный продукт, финальное тестирование продукта, приемочное тестирование и передача заказчику и, наконец, сопровождение и вывод из эксплуатации. В дальнейшем речь пойдет о практических приемах и более общих методах, которые необходимы для корректного завершения каждой из этих стадий в соответствии с различными подходами. При этом методология может включать применение моделей, методов и средств. Модели — более формальное представление

элементов или этапов, необходимых для реализации действий по разработке ЖЦ программных систем. Это прежде всего математические или другие формальные модели, скажем, модель виртуальной машины, функционирующей в среде Microsoft .NET, во многом основана на абстрактной машине, разработанной Юрием Гуревичем (специалист Microsoft Research). Модель носит формальный характер — она является математической моделью, основанной на понятии «состояние».

Кроме того, методология может включать методы, т.е. техники, которые являются менее формализованными. Одним из примеров может являться подход Microsoft Solution Framework, который содержит так называемые вехи (milestones) и результаты (deliverables). Кроме того, методология может включать (и в случае корпоративных систем, как правило, включает) применение специфических инструментальных средств, которые поддерживают весь жизненный цикл ПО. Это и анализ и разработка требований, и проектирование, преимущественно в форме диаграммирования, составления различных UML-диаграмм, кодирование и тестирование, Microsoft использует целый ряд специальных средств тестирования при реализации подхода MSF — реализация, отладка. Одним из примеров является Microsoft Visual Studio.NET, также поддерживается командная работа на основе Teamsystem или Teamsuit. Примерами классов таких средств являются средства быстрого прототипирования (rapid application development), CASE-средства компьютерной поддержки и разработки программного обеспечения или автоматизированной поддержки разработки ПО. Системы управления корпоративным контентом и целый ряд классов других систем.

Продолжим описание методологий разработки информационных систем и попробуем сосредоточиться на их пригодности — пригодности рассматриваемых классов методологии разработки информационных систем в отношении корпоративных систем. Если говорить о международных стандартах или методологиях, то это прежде всего IDEF-диаграммы и подходы, связанные со стандартом ISO. Федеральные российские стандарты — это стандарты ГОСТ и ESPD. В НИУ ВШЭ есть внутренний стандарт для производства документации, он достаточно четко отслеживается, даже при создании студентами курсовых проектов документация готовится в этих форматах.

Существует также целый ряд корпоративных стандартов, которые используются иногда несколько шире, чем предполагают пределы этих корпораций, — Rational Unified Process (RUP), который используется в и за пределами IBM, MSF, используемый преимущественно в Microsoft. Есть подход, который используется внутри корпорации Oracle, — CDM (Custom Development Method), который тоже во многом является корпоративным и вне стен Oracle, как правило, не используется.

Перечисленные подходы RUP, MSF, CDM можно отнести к корпоративным: они достаточно всеобъемлющи, широки и действительно охватывают полный жизненный цикл программных систем корпоративного типа, вполне применимы и по качеству подготовки документации, и по характеру и масштабу процессов для получения полномасштабных корпоративных информационных систем. Другие подходы, такие как Agile, eXtreme Programming (XP), Scrum, являются в некотором смысле ограниченными, в частности потому, что не всегда поддерживают полномасштабную документацию, и выход по проекту в полном смысле этого слова не может быть назван корпоративным программным решением. Эти подходы хороши для проектов с большой неопределенностью, которые характеризуются высокими рисками, когда изначально традиционные методологии, перечисленные в разделе корпоративных, могут не вполне адекватно работать. На самом деле нет гарантии, что сработает и один из этих подходов, но все же они разрабатывались специально для того, чтобы вести такие высокорисковые, сложные и неопределенные проекты. Конечно, в полном смысле такие подходы, как Agile, XP, Scrum, нельзя назвать корпоративными. Они не приводят к решениям корпоративного типа[1].

Таким образом, существует целая иерархия подходов к разработке систем. При этом то, что называется моделями ЖЦ (каскадная, спиральная модель) и методологии (такие как RUP, XP) — это во многом параллельные направления разработки корпоративных информационных систем. То есть работая в рамках RUP или, скажем, MSF, можно вести проектирование ИС по спиральной или каскадной модели. Эти понятия не являются взаимоисключающими, скорее они дополняют друг друга. В этой связи модели и методо-

[1] Более подробно см.: *Зыков С.В.* Проектирование интернет-порталов. М.: МФТИ, 2005.

логии являются понятиями ортогональными. Остановимся на тех методологиях, которые представляют основной интерес с точки зрения проектирования информационных систем и применимости для корпоративных ИС.

Первые подходы — это ГОСТ, ISO, т.е. стандарты. Это достаточно всеобъемлющий список документов, которые призваны поддерживать процессы проектирования и разработку программных продуктов корпоративного типа. Однако в практике проектирования часто это идет вразрез с интересами и требованиями заказчика, т.е. часто проектирование и подготовка полномасштабной документации по ГОСТ и ESPD являются избыточными, и западные стандарты ряд документов не поддерживает или поддерживает в ограниченном объеме.

В следующих главах будут более подробно рассмотрены RUP, MSF, CDM и гибкие методы Agile, XP, Scrum, которые в определенном смысле и в определенной степени могут применяться для корпоративных систем и при этом являются достаточно прагматичными. Если говорить о RUP, он может включать как каскадный, так и спиральный вариант проектирования с точки зрения модели жизненного цикла, но в целом он основан на итеративном подходе и включает быстрое прототипирование. Быстрое прототипирование, в принципе, можно выделить как модель жизненного цикла, но эта модель не является самостоятельной — она не поддерживает разработку боевого кода программной системы, т.е. не позволяет получить достаточно хорошо документированный и надежный код с точки зрения работоспособности и количества ошибок. Кроме того, этот код недостаточно масштабируем, он не рассчитан на большое количество одновременных пользователей и на те функциональные ограничения по количеству пользователей, по пропускной способности сети, по нагрузке на серверы программного обеспечения, по работе с базами данных, которые будут испытывать полномасштабные версии корпоративной информационной системы. Поэтому быстрое прототипирование достаточно хорошо как дополнительный подход, метод и модель жизненного цикла, который применяется в рамках RUP вместе с итеративным подходом. Этапы жизненного цикла здесь называются потоками. В явном виде выделяются роли. Ниже будет подробнее изложено об этом и о том, как производится документация, какие артефакты процессов, связанных с RUP, важны для ИС, корпоративных ИС.

Другой подход связан с синтезом каскадной и спиральной моделей — это MSF. Важные аспекты этого подхода — синхронизация и стабилизация. В основе проектирования программного обеспечения по этой схеме лежит процессный подход. Процессы, активности будут описаны подробнее в главе, которая будет посвящена этой тематике.

Еще один менее известный и используемый подход, более жесткий с точки зрения детерминированности и определенности этапов ЖЦ и связанный с каскадной моделью преимущественно — это Oracle CDM. Он используется для производства программных систем, в том числе и корпоративных программных систем, на основе продуктов Oracle — это Oracle Enterprise/Database Server, Oracle Business Suit, семейство модулей, которые предназначены для ERP, учета, планирования и управления корпоративными ресурсами: людскими, финансовыми и производственными ресурсами, прошлым документооборотом и целым рядом других ресурсов. При внедрении Oracle Applications сейчас вполне может использоваться этот подход. Также важно, что он включает прототипирование, это позволяет облегчить и удешевить процессы проектирования.

Гибкие методологии, о которых мы будем говорить отдельно, — Agile, XP, Scrum. Они основаны на итеративном подходе к ЖЦ, т.е. последовательном уточнении программного продукта по мере согласованием с пользователем требований к нему. Поскольку продукты, которые разрабатываются в рамках этих методологий, имеют изначально достаточно высокую степень неопределенности, в этой связи важно понятие рефакторинга, или последовательного улучшения кода. Также достаточно распространенное применение получили так называемые лучшие практики, или некоторые неформальные критерии и приемы разработки программного обеспечения. Неформальные потому, что сложно разработать количественные методы оценки этих критериев и в ряде подходов эти практики могут использоваться как в полном объеме, так и в некотором подмножестве. Эти подходы наиболее гибки.

Нужно заметить, что и MSF, и RUP имеют некий диапазон возможных вариаций для разработки программных систем того или иного класса и масштаба. То есть, в принципе, можно использовать MSF и RUP не только для корпоративных систем, но и с некоторыми ограничениями и упрощениями — для систем меньшего класса. Для этого также существуют специальные ограничения,

специальные подходы — так сказать, урезанные, сокращенные методологии. Такие подходы позволяют сэкономить на жизненном цикле, на производственном процессе как по времени, так и по людским ресурсам, а в итоге — по стоимости.

Завершая рассказ о введении в корпоративные информационные системы, нужно сделать следующие основные выводы. Понятие системы возникает, когда речь идет о корпорации — большой, территориально распределенной производственной структуре с общими бизнес-задачами, но с различными направлениями деятельности, с различными языками реализации, т.е. требуется локализация для разных стран тех систем, которые внедряются. Вообще говоря, программное обеспечение, которое производится для корпорации, представляет собой комплекс систем, которые нацелены на анализ, учет, планирование и управление различными областями деятельности этой корпорации. При этом такой комплекс имеет достаточно сложную схему взаимодействия. Одним из возможных решений по объединению такого рода систем является корпоративный портал. Эти программные компоненты создаются на основе различных архитектурных подходов. Это могут быть мейнфреймы, системы на основе архитектуры файл—сервер, клиент—сервер, интернет-архитектуры, различных технологий баз данных, например Oracle, Microsoft и т.д. Это могут быть системы, которые хранят информацию различной степени структурированности: хорошо структурированные реляционные таблицы, слабо структурированные аудиовидеоданные, отсканированные документы с нечетко определенными полями и т.д. Поэтому схемы взаимодействия элементов этого комплекса достаточно сложны. И для того чтобы понимать важность этих задач, необходимо представлять себе сложность. Это терабайты информации, в ряде случаев — уже петабайт. Так, скажем, информационные системы корпорации Intel в своей совокупности представляют уже несколько петабайт, т.е. крайне большой объем информации. В этой связи корпоративные информационные системы представляют собой достаточно сложный объект для исследования. Такие системы достаточно быстро растут: за пять лет объемы данных примерно удваиваются. То есть можно говорить о быстром росте объемов данных, в этой связи еще сложнее становится управлять такими большими программными комплексами.

Это только некоторые аспекты. Вообще разработка информационных систем — это многоаспектный процесс. В течение разра-

ботки информационная система проходит целый ряд стадий, связанных с так называемым жизненным циклом. Это и анализ и спецификация требований, и проектирование — первичное и детальное, и реализация, тестирование, интеграция, передача заказчику в эксплуатацию, сопровождение. То есть это достаточно сложный процесс последовательно сменяющих друг друга стадий. Тем не менее этот процесс является замкнутым и итерационным: ряд стадий при отдельных подходах, скажем, при спиральной, эволюционной, инкрементной моделях, последовательно повторяется. Конечно, для того чтобы говорить о корпоративных системах, нужно достаточно детально и полно представлять себе все этапы этого жизненного цикла, взаимодействие этих этапов и ту документацию, которая производится на каждом этапе и является во многом основой такого (в том числе документального) взаимодействия. При каскадном подходе нельзя перейти к следующему этапу, если предыдущий этап документально не закрыт и на соответствующем документе, который подтверждает корректность и финализацию этого этапа, не появляется подпись ответственного лица. В этой связи нужно представлять себе всю схему ЖЦ, для того чтобы корректно выбрать модели, методы и средства, которые включают в себя те методологии проектирования КИС, о которых мы будем говорить дальше.

Остановися подробнее на жизненном цикле ПО (ЖЦ ПО). В начале главы уже были упомянуты его основные этапы. Фазы, которые связаны с разработкой программных систем, включают: анализ и спецификацию требований к программному продукту, проектирование программного продукта — первичное и детальное, уточненное, реализацию и тестирование элементов или модулей отдельного программного продукта, интеграцию или сборку этих модулей в частичный или полный программный продукт вместе с интеграционным тестированием, передачу заказчику после приемочных тестов, промышленную или опытную эксплуатацию, которая называется сопровождением и занимает по времени и средствам основную часть жизненного цикла, и, наконец, вывод из эксплуатации. Для реализации этого жизненного цикла применяются различные модели, методы и инструментальные средства. Подробнее рассмотрим основные этапы ЖЦ.

Прежде всего определим, что такое ЖЦ и в чем состоят его особенности для систем корпоративного типа. Ведь речь идет о

действительно больших системах, которые включают терабайты данных разных степеней структурированности, географически распределены по земному шару и между которыми нужно наладить взаимодействие для получения консолидированной отчетной информации по основным видам корпоративных ресурсов. Будут рассмотрены основные этапы ЖЦ: анализ и спецификация требований, эскизное и детальное проектирование, реализация и тестирование, сопровождение и вывод из эксплуатации — и экономическая специфика этапов ЖЦ ПО. При этом будет упомянуто не только о стоимости затрат, но и об их структуре, на основе анализа большого количества проектов, который был произведен в частности компанией HP и другими компаниями, здесь будут приведены оценки, сделанные Карнеги-Мелонским университетом. И, что очень важно, будет рассмотрена связь этапов ЖЦ с различными моделями.

Модели, методы и инструментальные средства — это, так сказать, три кита, три основных составляющих, на которых стоит все проектирование, разработка корпоративных, в том числе информационных, систем. Эффективная разработка немыслима без использования средств автоматизированного проектирования, или CASE-средств. При описании производства как промышленного процесса необходимо упомянуть о тех метриках, которые позволяют определить и ограничить программный продукт и приходить к определенным выводам на основании анализа этих метрик. Это позволит ответить на вопросы: следует ли уже прекратить сборочное тестирование и начинать тестирование продукта, достаточно ли качественным является этот продукт, не превосходит ли существующее количество ошибок некое пороговое значение.

Как оценить сложность ПО? Достаточно ли, скажем, для этого ограничиться количеством строк кода? Или существуют другие метрики оценки? Например, количество операторов, операндов и т.д. Как пользоваться этими метриками и насколько они эффективны? Ведь процесс производства ПО должен быть конвейерным, таким, чтобы методологии и модели работали для большого количества программных продуктов и проектирование проводилось по единообразной схеме.

Итак, в чем заключается жизненный цикл программного обеспечения, какие имеются у него составляющие, в чем состоит его экономика, инструментарий и метрики.

В разработке ПО существуют определенные сложности и проблемы, которые нужно решать со стороны как разработчика, так и системного архитектора и даже руководителя программного проекта. Необходимо достичь определенного уровня качества, которое связано с теми метриками, о которых уже было сказано: порог ошибок, интерфейс пользователя, эргономичность, масштабируемость, количество одновременных пользователей, количество транзакций, время реакции системы и объем БД. Программный продукт должен удовлетворять этим метрикам при определенном дефиците ресурсов, имеющем место в любом проекте и который в любом случае достаточно жестко контролируется в ходе программных проектов. Это возрастающая сложность программных систем. Корпоративные системы — это десятки взаимодействующих систем, каждая из которых объединяет зачастую сотни первичных сущностей и часто терабайты данных, т.е. это очень сложные программные системы, которые достаточно быстро растут и которым необходимо взаимодействовать друг с другом.

В ходе выполнения программных проектов приходится часто сталкиваться с нехваткой ресурсов: людских, временны́х, финансовых. Происходит постоянное взаимодействие с заказчиком, который часто изменяет требования, и в ряде случаев, эти изменения могут носить для разработчика достаточно сложный и плохо предсказуемый характер, иногда эволюционный, иногда революционный. При этом необходимо, в зависимости от пути или степени изменения этих сложностей и требований, корректировать модели ЖЦ и, соответсвующим образом, очередность стадий ЖЦ программных продуктов.

Еще один важный аспект — это проектная команда, взаимодействие большого количества участников. Под участниками в ряде случаев понимаются представители не только разработчика, но и заказчика, которые входят в состав software quality assurance — группы контроля качества продукта. Если даже исключить их из рассмотрения, а в ряде методологий, в особенности гибких (Agile, XP, Scrum), эти представители присутствуют и играют достаточно активную роль, то в любом случае на стороне разработчика есть целая проектная команда (может быть не одна), работу которой нужно координировать. В больших программных системах это большой объем человеко-часов и большое количество исполнителей с разными мотивациями, целями и задачами. В этом смысле, при большом количестве

участников, необходимо управлять процессом, привлекая к этому CASE-средства (автоматизированного проектирования) — это тоже достаточно сложно. При этом важной проблемой является моральное устаревание программного обеспечения.

В следующих главах будет подробнее изложено о понятии Software Engineering (программная инженерия), которое возникло в конце 1960-х гг. на конференции NATO, когда обсуждалась аналогия между любым процессом промышленного производства (в частности, строительством мостов) и строительством программного обеспечения, программной архитектурой. Вообще достаточно часто в литературе, связанной с ПО, возникают аналогии между архитектурным строительством сооружений, зданий, мостов и программными проектами. В отношении Software Engineering — тут не все так просто. Ряд методов, которые работают в первом случае, не подходят для программной инженерии. Программное обеспечение морально устаревает — и это происходит достаточно быстро. Посмотрим, например, на скорость смены ОС Windows — это происходит примерно раз в 5 лет, может и чаще. В то же время многие дома и мосты морально не устаревают гораздо дольше, в течение сотен лет. Таким образом, проблемы разработки ПО во многом более динамичны, чем проблемы целого ряда отраслей реального сектора экономики. Кроме того, разработка ПО растет высокими темпами. Для ряда компаний это направление является единственным, основным, определяющим. Необходимо успеть до того, как выйдут на рынок продукты конкурентов, опередить их и обеспечить высокое качество продукции, совместив его с достаточно быстрым вводом в эксплуатацию. Кроме того, это очень большое количество новых отраслей народного хозяйства. Достаточно сказать о такой новой отрасли, как нанотехнологии — это очень быстрая, конкурентная отрасль, которая затрагивает целый ряд промышленных технологий и направлений, требует оперативного знакомства предметных экспертов и системных аналитиков с совершенно новыми понятиями. Таким образом, получается достаточно большое количество взаимосвязанных и взаимодополняющих проблем, которые существенно осложняют разработку ПО, особенно в корпоративных системах.

Графическое представление ограничений на разработку приложений можно описать следующим образом. Это некоторая модификация традиционного проектного треугольника, который

связан с затратами времени, средств и функционала. Приблизительно можно увидеть это как три оси. Где-то внутри этого треугольника находится оптимальное сочетание этих параметров, которое и удается обеспечить при адекватном сочетании моделей, методов и средств проектирования корпоративных информационных систем и программных приложений в целом.

Какие ограничения можно увидеть при разработке приложений, в том числе корпоративных? На процессы разработки воздействует целый ряд факторов. Это, конечно, объем кода, который можно измерить в тысячах строк. Корпоративные продукты — это десятки, сотни тысяч строк и более, в зависимости от характера и масштаба этих систем. Это, конечно, очень большая сложность. Каждый отдельно взятый модуль таких систем, как, например, Oracle Applications, представляет собой несколько сотен первичных сущностей. Для того чтобы охватить их взглядом, требуется очень серьезная фундаментальная предметная подготовка, аналитический взгляд, профессионализм и использование специализированных средств автоматизированного проектирования. Кроме того, существует целый ряд ограничений, которые связаны с людскими ресурсами. Естественно, человеку охватить такое количество сущностей и грамотно строить процессы проектирования, разработки, которые включают и тестирование, постановку задачи, анализ и спецификацию требований, естественно, очень сложно. Эти процессы нужно грамотно координировать, чтобы команда давала отдачу от того, что используется такой большой коллектив, и не появлялись чрезмерные затраты на обучение все новых и новых членов команды по мере того, как проект расширяется и в него вовлекаются новые силы и средства.

Кроме того, обучение тормозит процессы разработки. Нужно понимать, что в разработке новой системы в рамках существующего корпоративного программного комплекса заказчик зачастую не может остановить свои ключевые бизнес-процессы. И пусть ценой дополнительных затрат унаследованные системы, работающие на мейнфреймах или устаревших архитектурах, все же обеспечивают поддержку этих бизнес-процессов. И при интеграции новых систем в существующую программную среду заказчика неоходимо обеспечить корректность и адекватность этой интеграции и функционирование расширенного комплекса новой системой. Это нужно сделать для того, чтобы этот новый комплекс позволял извлечь

больше информации, консолидировать данные и в итоге давал возможность руководству получить аналог приборной панели, на которой оно сможет видеть результаты, достигнутые компанией по финансам, по кадрам, по материальным и производственным ресурсам. Результаты можно будет представить в виде срезов, проекций с детализацией до отдельных стран, компаний, подразделений и сотрудников, и это позволит плавно масштабировать и анализировать эти результаты, строить тренды, прогнозы перспектив развития.

Целями разработки являются снижение или оптимизация ресурсов — многофакторная оптимизация, которая связана с людскими ресурсами, оптимизацией стоимости и длительности времени графика, плюс функциональные ограничения — ограничения на ту функциональность, которую необходимо и желательно реализовать в рамках программного проекта.

В чем состоит современный подход к решению всех этих проблем? Проблемы, связанные с важностью, высокотехнологичностью и ограничениями, которые диктует рынок: конкурентная среда, время, которое жестко ограничивает регламенты проектирования корпоративных систем, и ряд других проблем, которые усугубляются корпоративным характером информационных систем, сложностью и большим объемом приложений. Конечно, можно прибегнуть к методам анализа и систематизации тех знаний, которые уже существуют, и которые были получены имперически при разработке первых подобных проектов, подобного класса и масштаба. И здесь на помощь приходит программная инженерия, то есть целый ряд дисциплин, которые связаны с процессами управления проектированием программных систем, построением архитектурных основ такого рода информационных систем и, конечно, разработки, проектированию и реализации, включая тестирование и сопровождение, то есть управление ЖЦ такого рода программных систем и их комплексов.

Глава 2
ОБЗОР ЖИЗНЕННОГО ЦИКЛА КОРПОРАТИВНЫХ СИСТЕМ

Программная инженерия, или инженерия программного обеспечения, представляет собой область компьютерной науки, которая занимается построением программных систем, т.е. целого ряда взаимодействующих компонентов программного обеспечения, которые являются настолько большими или сложными, что для построения такого рода систем требуется участие команды или даже взаимодействующих команд разработчиков. Под разработчиками здесь понимаются не только программисты, но и аналитики, постановщики задач, тестировщики, системные архитекторы, документаторы, специалисты по контролю качества ПО и персонал сопровождения. То есть это достаточно большая команда, которая нацелена на производство того или иного программного продукта в уже существующей среде информационных систем заказчика. Поэтому очень важен подход к организации на всех уровнях и во всех перечисленных аспектах разработки программного обеспечения: анализ и спецификация требований, первичное и детальное проектирование, реализация и тестирование, интеграция, передача заказчику, сопровождение.

Программная система — это совокупность взаимодействующих программ под общим управлением, которая предназначена для того, чтобы решать конкретную задачу или ряд взаимосвязанных задач.

Приложение — это программа, которая решает функциональные задачи по обработке информации в рамках той или иной предметной области, например приложения, которые контролируют людские или другие ресурсы.

Процитируем В.А. Липаева — патриарха отечественной программной инженерии. В книге «Программная инженерия» он привел следующее определение: «Под программной инженерией понимается комплекс задач, методов, средств и технологий создания, то есть проектирования и реализации сложных, расширяемых, тиражируемых, высококачественных программных систем, возможно включающих базы данных»[1]. Каждое слово в этом определении в полной мере применимо к корпоративным системам.

[1] *Липаев В.В.* Программная инженерия. М.: ТЕИС, 2006.

Согласно определению Липаева эта отрасль науки как раз и направлена на создание корпоративных информационных систем. И ввиду того, что они являются сложными, т.е. содержат большое количество первичных сущностей, большими по объему (тера-, петабайты данных) и расширяемыми, как правило, речь не идет о том, что мы революционным образом сразу заменяем все системы, которые используются в корпоративном программном комплексе. Чаще всего производится доработка какой-то отдельной системы. Они являются высококачественными и часто тиражируемыми.

Некоторые из примеров таких решений — Microsoft Dynamics, Oracle Applications и т.д. Под высоким качеством понимается и масштабируемость — плавное снижение производительности при достаточно резком увеличении интенсивности нагрузки на систему. Кроме того, нужно сказать, что эти системы должны быть надежными, вести себя предсказуемо, быть эргономичными, сопровождаемыми, т.е. должны быть настроены на то, чтобы обеспечивать достаточно гибкое и относительно эволюционное взаимодействие с пользователем на этапе опытной и промышленной эксплуатации. В определении также речь идет о проектировании и реализации, т.е. уже о полном жизненном цикле ПО. Важным дополнением является то, что информационные системы включают в ряде случаев базы данных. Если мы говорим о корпоративных системах, базы данных, как правило, являются неотъемлемой, важной частью этих систем. Другое дело, что эти базы данных могут строиться на различных принципах, являться гетерогенными, включать объектные составляющие, т.е. быть не чисто реляционными. Последние версии СУБД Oracle называются объектно-реляционными. Есть СУБД нового поколения, такие как O2, Orion и др., которые используют не только реляционные, но и другие, более новые объектно-ориентированные модели.

В корпоративных информационных системах необходимо разделять понятия программного проекта и программного продукта. В настоящем издании речь пойдет в основном о программных продуктах, т.е. о взгляде на ЖЦ с точки зрения системного архитектора. А с точки зрения программного проекта — это взгляд менеджера проектов, когда речь идет об управлении проектной командой, проектами, взаимодействием людей в проекте, сроками, стоимостью. С другой стороны, если говорить о программной инженерии, то речь может идти о разработке продукта для конкрет-

ного заказчика, но преимущественно и предпочтительно планировать все процессы и технологии, связанные с разработкой таким образом, чтобы по возможности обеспечить производство продукта, нацеленного на более широкую аудиторию потребителей, а в идеале сделать его коробочным или тиражируемым. Целесообразно обеспечить высокий процент повторного многократного использования элементов проекта — это и код, и документация, и структура СУБД, и программная архитектура, с тем чтобы при доработке проекта для «похожего» заказчика было затрачено минимум времени, средств и людских ресурсов.

На начальной стадии разработки продукта, как правило, речь идет о концепции, о том, что существует идея. При этом, конечно, необходимы начальные инвестиции. В то же время, если говорить о разработке проекта, то здесь существует уже некоторый черновой план, который учитывает основные финансовые, функциональные и временны́е ограничения, есть заказчик и конкретные лица, которые могут обеспечить финансирование проекта. В ряде случаев речь может идти о смешанной разработке, когда разработка частной системы под конкретный заказ может трансформироваться в относительно открытое решение для широкого класса заказчиков.

Чем характеризуется программный продукт? Во-первых, как правило, он имеет определенную коммерческую ценность. Это значит не то, что не существует условно бесплатных программных продуктов, а что этот продукт решает конкретную задачу конкретного класса пользователей, потребителей продукта. Таким образом, продукт может называться рыночным и быть предложен рынку для удовлетворения его определенных потребностей и решения конкретных бизнес-задач. Какие примеры программного продукта можно привести? Часто это физические объекты, скажем, информационные носители (DVD, CD и т.д.), но это могут быть и нематериальные соглашения, такие как лицензия, соглашение о партнерстве и пр. Еще одним примером может выступать услуга по внедрению, сопровождению, консалтингу и т.д.

Программные продукты можно классифицировать по разным основаниям. Один из видов классификации — масштаб использования: это и личное использование, и некоммерческое, и коммерческое как коробочный продукт для широкого класса организаций и предприятий. Другой способ классификации — цель использования. Это может быть специализированное ПО, нацеленное на

решение достаточно узкой задачи, например расчетное ПО для решения астрономических задач, лазерной дальнометрии, или ПО более общего назначения, такое как операционная система, офисные продукты и т.п. Еще один вид классификации — степень открытости. При этом можно говорить о компонентной ориентированности, скажем, API, библиотеки, такие как, например, библиотека классов Enterprise Libraries, которая используется для надстройки над .NET для построения Microsoft-продуктов корпоративного типа, библиотека классов для построения офисных приложений и т.д. или готовые продукты.

Любая разработка ПО происходит согласно жизненному циклу и включает последовательное прохождение стадий ЖЦ, о которых мы уже упоминали и которые в широком смысле начинаются с концепции или базовой идеи и заканчиваются выводом из эксплуатации.

Вообще говоря, понятие ЖЦ можно использовать и применительно к другим классам систем, например, к таким системам, как архитектурные сооружения, однако ЖЦ программных систем (ПС) имеет свои особенности. ПС разрабатывается постепенно и развивается от концепции, абстрактной идеи и далее конкретизируется до программного продукта, который включает в себя не только код, но и большое количество документации — это диаграммы, документация к коду, документация для пользователей по работе с программным продуктом, для администраторов по настройке, установке и сопровождению и т.д. Программные системы заканчиваются на этапе вывода из эксплуатации, который завершает сопровождение.

Каждый этап ЖЦ завершается разработкой некоторой части системы — она может быть полнофункциональной или не совсем полнофункциональной. Это зависит от конкретной модели ЖЦ. Каждый этап завершается производством документации, которая включает более глобальные артефакты, такие как план проекта, план тестирования, план реализации, план сопровождения, или более узкие документы, такие как сценарии использования, руководство администратора, краткое руководство пользователя, основные требования к проекту или более детальные требования в форме технического задания. Объем, характер и масштаб документации зависят от характера и масштаба программного продукта. Конечно, для каждого этапа производства ПО должны быть четко

определены начальные и конечные временные точки, а также известны элементы, которые должны быть переданы следующему этапу, с точки зрения кода, документации, базы знаний. На практике все обстоит сложнее, но в любом случае, если мы говорим о технологии проектирования ПО, то каждый этап ЖЦ с определенностью завершается производством некоторого нового продукта и новой документации к нему.

Изучение жизненного цикла корпоративных программных систем необходимо прежде всего для понимания организации разработки ПО, т.е. всех процессов, которые связаны с ЖЦ. Не поняв того, как устроен ЖЦ вообще, нельзя говорить о сколько-нибудь планомерной организации и управлении этими процессами. Конечно, из успешных проектов нужно делать выводы и тиражировать принципы, которые привели нас к успеху, практические шаги и методы, которые дают возможность эффективно и планомерно развивать проекты, совершенствовать программные продукты, взаимодействие с пользователями, производство документации и все процессы, которые лежат в основе жизненного цикла. Это необходимо делать на основе анализа результатов работы над предыдущими проектами, и тут может помочь и план проекта, и другие глобальные документы (план тестирования, интеграции, реализации, сопровождения), а также вся проектная документация, которая была создана, а кроме того, журналы ошибок и документы, создающиеся на этапе сопровождения программного продукта. В этом смысле изучение ЖЦ дает важную основу для анализа разработки ПО, которая позволяет более тщательно планировать и производить процессы, лежащие в основе ЖЦ, и тиражировать таким образом внутреннюю методологию, которая будет разработана и развита командой разработчиков. Так можно прийти к корректной и адекватной постановке и адаптации. Конечно, для каждого программного проекта, как правило, приходится иметь дело с некой стандартной методикой, методологией, в том числе с учетом всего предыдущего опыта, и с необходимостью адаптировать его с учетом характера и масштаба проекта к конкретному заказчику и конкретным условиям производства продукта. Важно помнить, что у заказчика имеется определенная и, как правило, уникальная комбинация программно-аппаратной среды, в которую предстоит интегрировать программный продукт. Это особенно важно применительно к корпоративным системам, поскольку обусловливает боль-

шое количество взаимосвязей и значительный объем и сложность этого программного окружения и в целом корпоративного программного комплекса. Таким образом, анализ ЖЦ — необходимая стадия для крупных и сложных программных проектов, каковым является КИС.

При разговоре о ЖЦ необходимо сделать ряд важных замечаний. Прежде всего, нужно сказать, что процесс ЖЦ как создания, так и смены стадий ПО включает целый ряд сторон. Как минимум, это представители заказчика, представители разработчика и руководство. При этом представители заказчика — те, кто будут принимать продукт, во многом технически грамотные люди. В итоге они входят в группу контроля качества программного продукта. Со стороны разработчика — это весьма широкий спектр специалистов: аналитики, оценщики рисков, проектировщики, системные архитекторы, документаторы, программисты, тестировщики, специалисты по созданию приемочных тестов, специалисты по сопровождению. И руководство, которое разделяется, например, как в MSF, на руководителя проекта и руководителя продукта. То есть у руководства тоже различные цели, не говоря уже о том, что у руководства заказчиков и руководства разработчика они во многом расходятся.

У этих разных сторон зачастую совершенно различные цели и ожидания от продукта, от того, какую функциональность он должен реализовывать, и от проектных ограничений и по срокам, и по стоимости, и по функциональности и даже различное понимание определенных терминов и особенностей. Ведь заказчик смотрит на процесс производства ПО, достаточно хорошо понимая свои производственные потребности, но он может не вполне владеть особенностями производства ПО, которые приняты у разработчиков. И в этой связи даже разумные взгляды, подходы и отношения к ЖЦ, к тем требованиям и ограничениям, которые имеются у заказчика и разработчика и различных представителей заказчика и разработчика на различных уровнях, могут приводить к значительному увеличению сроков и стоимости проекта. Большое значение имеет согласование этих подходов между разработчиком и заказчиком: нужно прийти к некоему общему пониманию, прежде всего ограничений проекта. И надо помнить, что заказчик тут стремится ограничить проект снизу, заявить о том, что количество пользователей не может быть меньше, чем некое число, и т.д. А разработчик должен прийти к соглашению с заказчиком (к юридичес-

кому документу, техническому заданию, списку-требованию или какому-то иному документу). Разработчик при этом стремится ограничить проект сверху (количество пользователей, пропускная способность), т.е. показать, что технологии, которые используются, и бюджет, который заложен в проект, не могут обеспечить производительность более установленного предела.

Выше были перечислены некоторые участники проекта: руководитель портфеля проекта, менеджер проекта, руководитель команды, эксперт в предметной области, предметный аналитик, другие классы аналитиков, системный архитектор, проектировщик подсистемы или модулей ПО, специалисты по пользовательскому интерфейсу, в том числе по его тестированию, созданию его эргономики, кодировщики, сборщики, тест-менеджеры (создатели юнит-тестов, модульных, приемочных, сборочных тестов), тестировщики, руководители групп тестирования, технические писатели и целый ряд других. Эти роли обозначают только классы участников проекта, а классы конкретизируются в крупных проектах большим количеством участников. Взаимодействие между ними — это достаточно сложная задача с точки зрения управления и проектом, и продуктом. В дальнейшем мы будем говорить преимущественно об управлении продуктом.

Какой же целью задаются разработчики? Главная цель — это создать хороший продукт. (Что такое «хороший», будет расшифровано далее, а также какие именно факторы разработки ПО должны в первую очередь приниматься во внимание.)

Следует напомнить, что производство ПО представляет собой многофакторную оптимизацию, поскольку, по сути, разработчикам необходимо согласовывать с заказчиком некий взгляд и набор требований к проекту. Это будет основным сырьем, по которому будет создаваться программный продукт, включающий документацию. При этом выход по программному обеспечению может быть множественным, потому как очень часто приходится сталкиваться с ситуацией, когда существует огромное количество вариаций кода, которое решает поставленные перед разработчиком задачи. При этом, если говорить о ЖЦ ПО, следует нужным, предсказуемым и правильным образом с точки зрения сроков, стоимости и функциональности обеспечить выбор методологии этого ЖЦ. Необходимо показать, каким образом будут меняться фазы и сколько раз, сколько итераций нужно будет для того, чтобы получить продукт

должной функциональности, сложности и качества. При этом производится многомерная, многофакторная оптимизация, которая учитывает прежде всего следующие параметры: сроки выполнения проекта, стоимость продукта, качество продукта как по документации, так и по коду. Качество документации можно отслеживать трассировкой документации, сопоставлением артефактов, или элементов документации, на внутреннюю корректность, на соответствие друг другу, на полноту, на непротиворечивость, на целостность и на соответствие исходной постановке задачи. Также важным фактором оптимизации является сопровождаемость — обеспечение сокращений затрат на самую ресурсоемкую часть ЖЦ продукта. Важно сказать, что приоритетность факторов не жестко детерминирована, а во многом определяется характером и масштабом программного проекта. О каких масштабах имеет смысл говорить в отношении корпоративных программных систем? Для малых систем масштаб условно можно ограничивать 10 человеко-годами, для средних систем — 10–100, для больших — 100–1000 человеко-лет. Несколько тысяч — это уже огромные системы. Корпоративные системы — скорее от 100 человеко-лет и выше. То есть это весьма большие затраты, но это не означает, что не нужно искать возможности для экономии. Конечно, это нужно делать, и в первую очередь можно сэкономить гораздо существеннее на внедрении корпоративного приложения.

Нужно сказать, что продукт и проект — это различные понятия и, вообще говоря, те стадии, которые учитывают ЖЦ продукта несколько шире и включают в себя оценку возможности создания этого проекта и концептуальную основу проекта, идею, с которой он начинается. ЖЦ проекта во многом завершается при передаче в эксплуатацию каждого конкретного релиза этого продукта. ЖЦ продукта включает и сопровождение, и эксплуатацию, и вывод из эсплуатации.

Если говорить подробнее об экономике ЖЦ программного продукта, то нужно сказать, что он проходит целый ряд стадий и эти стадии вносят различный вклад в прибыль и динамику продаж. И если на стадии создания и вывода на рынок проект преимущественно находится в минусе по прибылям, то после вывода на рынок, когда наблюдается рост и зрелость, прибыль становится положительной. В период зрелости, как правило, прибыль имеет отрицательный прирост, но положительное значение. В районе

упадка наблюдается уже существенное падение и невысокое значение как прибыли, так и продаж с точки зрения их динамики. Все это характерно как для коробочного продукта, когда речь идет о количестве инсталляций, так и в случае, когда взаимодействие осуществляется с конкретным заказчиком, с учетом всех классов пользователей продукта.

Можно рассмотреть более подробно экономику ЖЦ на основе сопоставления критериев развития, скорости роста бизнеса и доли рынка, которую занимает программный продукт. Здесь, в начале пути, нужны инвестиции, поскольку неизвестно о дальнейшей судьбе программного продукта. Затем программный продукт выходит на рынок, приносит доход и, наконец, прибыль, и это без существенных затрат на поддержку продаж. Через некоторое время наступает этап, когда доходы относительно невысоки и продажи влекут за собой существенные затраты.

Если вернуться к описанию стадий ЖЦ ПО, то в ходе анализа можно выделить, что существует целый ряд стадий, которые практически не зависят от применяемых методологий разработки программных систем. Эти стадии включают анализ требований к программному продукту, подготовку проектных спецификаций программного продукта, проектирование (эскизное, первичное, детальное, окончательное, рабочее), реализацию, тестирование (модульное, компонентов), интеграцию (вместе с тестированием), сопровождение, вывод из эксплуатации.

Важной составляющей продукта является не только код, но и документация. Достаточно распространено заблуждение, что документация не нужна или ею можно пренебречь. Документация — очень важный выход по программному продукту. Если вы является разработчиками или вам приходилось разрабатывать продукт, то достаточно вспомнить о вашем коде, который вы пытались читать спустя несколько лет после его создания. Наверное, вы помните, что это не очень легко без хорошей документации. На стадии сопровождения, когда приходится читать чужой код, что на самом деле не очень просто, и код читает человек, который имеет достаточно средний уровень знаний программирования, конечно, человеку сложно читать код, если он был вне этого проекта. Но, как правило, именно это и происходит. Становится понятно, что без документации читать такой код практически невозможно. Например, в корпорации Microsoft проектная команда собирается исклю-

чительно для создания программного продукта, и после этого, как правило, люди друг с другом больше не встречаются. Таким образом, поддержка кода осуществляется исключительно благодаря той документации, которая его сопровождает, поэтому роль документации очень велика, затраты на нее существенно окупаются, и только она обеспечивает гибкость и мягкость сопровождения. После сопровождения происходит снятие с эксплуатации. Еще очень важно: документация обеспечивает взаимосвязь этапов жизненного цикла. То есть те документы, которые производятся на этапе анализа требований, являются сырьем для подготовки проектных спецификаций, которые в свою очередь являются сырьем для проектирования. Документация по проектированию — это большое количество диаграмм, сценариев использования и пр., являющихся основой для реализации, и т.д. Таким образом, документирование является неотъемлемым атрибутом каждой стадии ЖЦ ПО.

Перечислим более подробно, что происходит на каждой стадии ЖЦ ПО. Первой стадией является *анализ требований*. При этом происходит встреча, как правило неоднократная, представителей разработчика и представителей заказчика. Целью является достижение общего понимания той самой задачи, на решение которой и будет направлено ПО, производящееся в интересах заказчика. Конечно, в ряде случаев заказчик может не обладать полнотой знаний о тех технологических особенностях ведения проекта, построения программного продукта, которые имеются у разработчика в том опыте проектной команды, тех технологиях, стандартах, которые применяются для проектирования, реализации и передачи заказчику. Очень часто заказчик может быть не вполне технически грамотным, но он имеет достаточно четкое представление о предметной области, в рамках которой должно быть произведено программное решение. С другой стороны, разработчик часто имеет ограниченное представление об особенностях той самой предметной области. Если говорить о нефтегазовой среде, например, то достаточно важным может быть представление результатов исследования сейсмической активности земной коры, в том числе в трехмерной динамике — трехмерное представление геологических данных о земной коре с учетом динамики. Это весьма специфический вид данных, который может не вполне адекватно восприниматься и анализироваться разработчиком, поскольку на стороне разработчика сложно найти специалистов в области геологии.

В других направлениях, например в угольной отрасли, геология имеет свою специфику, отличную от нефтегазовой отрасли. Этот пример показывает, что бывает достаточно трудно прийти к общему пониманию тех задач, особенностей, специфики, которые несет предметная область, для которой и реализуется программный продукт. Очень важно, что при этих встречах должно быть в полной мере выявлено и обсуждено все множество как функциональных, так и нефункциональных требований и ограничений заказчика на программное обеспечение, которое у него появится и будет решать его задачи, желательно в количественном виде. Это производится с помощью нескольких собеседований. В итоге появляется документ, который содержит формализованное описание требований к программному обеспечению в виде списка требований или технического задания. Этот результат имеет принципиальный характер, поскольку на основе требований, с учетом количественных ограничений и осуществляются последующие стадии (проектирования, реализации и т.д.) программного продукта.

Следующая стадия — *подготовка проектных спецификаций*. Она происходит на основе описания требований, т.е. тех документов, которые получены на предыдущей стадии ЖЦ. Эта и следующие стадии являются в основном прерогативой разработчика. Хотя в ряде методологий проектирования и реализаций программных комплексов, таких гибких, как Agile, XP, Scrum, заказчик участвует на всех этапах ЖЦ ПО. Для больших корпоративных систем, как правило, разработка ведется по методологиям RUP или MSF, и там основным действующим лицом является разработчик. Проектные спецификации содержат описания всей функциональности проекта и всех основных ограничений, желательно выраженных количественно. Здесь уже можно ограничить и программное обеспечение, и технологии, которые будут использованы, и архитектуру (например, сделать выбор между платформами Java или .NET). Необходимо четко ограничить количество одновременных пользователей, количество подключений, транзакций и их интенсивность, пропускную способность канала и ряд других параметров. При этом методологию или модель разработки ПО — каскадную, эволюционную, спиральную или иную — следует выбрать как можно раньше, поскольку выбор методологии или модели ЖЦ определяющим образом сказывается на сроках, стоимости и успехах проекта. Проектные спецификации должны ограничивать сроки и

стоимость проекта исходя из договоренностей, которые достигли разработчик и заказчик на предыдущем этапе.

Далее на основе проектных спецификаций производится детальное проектирование, которое описывает программную архитектуру с учетом всех компонентов проекта. В случае объектно-ориентированного подхода это модули и интерфейсы между ними, компоненты и средства их взаимодействия в условиях той программной среды, которой располагает заказчик. Однако в больших корпоративных системах всегда присутствует некоторое количество взаимодействующих систем, которые уже работают у заказчика, и, как правило, разработчики приходят к заказчику с предложениями, которые учитывают эти условия программной и аппаратной среды. У заказчика может быть множество серверов, например серверы баз данных, кэш-серверы, серверы безопасности, серверы, отвечающие за телекоммуникации, и пр. Детальное проектирование также выполняется разработчиком. Кроме написания программной архитектуры, детальное проектирование на выходе дает документы, которые описывают все программные модули корпоративного комплекса.

После детального проектирования и ревизии проекта, т.е. проверки спецификаций на внутреннюю корректность, полноту, непротиворечивость, целостность и на соответствие техническому заданию, можно переходить к реализации, т.е. созданию кода программного продукта и соответствующей документации.

Код программного продукта создается помодульно исходя из компонентов, которые были определены на предыдущем шаге. Реализация производится разработчиком на основе документов детального проектирования с учетом общего плана проекта, поскольку необходимо принимать важные решения об ограничении тестирования, сроках реализации индивидуальных модулей и переходе к интеграции и последующим стадиям, которые определяют успех передачи заказчику, с одной стороны, и качество программного обеспечения, с другой. Поэтому общий план проекта, который включает глобальные ограничения на сроки и стоимость, а также на важнейшие функциональные параметры и ограничения программного продукта, должен быть принят во внимание на этой стадии для обеспечения корректности, предсказуемости и качества процесса реализации. Реализация — это стадия, за которую отвечает разработчик, т.е. кодировщики, тестировщики. Разрабатыва-

ются отдельные модули — небольшие подсистемы, которые решают замкнутые задачи и для которых на предыдущем этапе уже заданы основные параметры, такие как алгоритмы и структуры данных, переменные — локальные и глобальные, основные (в случае ООП) структуры классов — их основные атрибуты и методы. В результате мы получаем отдельные программные модули, каждый из которых является уже реализованным и протестированным прежде всего самим разработчиком на внутреннюю корректность и на соответствие проектным спецификациям по отдельности. После реализации и на самом этапе реализации важными документами являются документы, связанные с тестированием, такие как: юнит-тесты, проектная документация к каждому модулю, краткое описание модулей, их назначение и интерфейсы, взаимосвязь с другими модулями, основные характеристики, атрибуты, методы, алгоритмы и структуры данных модуля, документация к коду, которая позволит достаточно легко читать и анализировать даже без запуска кода и без разработчика.

После производства отдельных модулей, когда принято решение о том, что они уже достаточно целостные, надежные и качественные, содержат некий порог ошибок, не превышающий максимального, можно переходить к этапу интеграции, т.е. к сборке в общую архитектурную схему, которая была оговорена на этапе архитектурного проектирования. Модули тестируются попарно, в совокупности образуя частичный и полный продукты. После чего разработчик и заказчик проводят финальное тестирование и происходит приемка программного обеспечения на основе приемочных тестов.

В первый раз ПО разворачивается у заказчика на его реальном программном и аппаратном окружении и реальных данных в тех объемах, которые определяются условиями эксплуатации корпоративных программных комплексов заказчика. Если все приемочные тесты, которые производятся заказчиком, успешны, т.е. продукт ведет себя надежно, корректно, соответствует функциональным требованиям, вписывается в программно-аппаратное обеспечение заказчика, то происходит передача программного продукта вместе с документацией заказчику и наступает фаза эксплуатации. Эта фаза относительно ЖЦ называется фазой сопровождения.

С точки зрения экономики сопровождение — самый затратрый этап ЖЦ (порядка $^{2}/_{3}$ стоимости всего проекта) как по времени, так

и по средствам. Нужно понимать, что сопровождение необходимо для любого ПО. Цель при разработке — не просто передача программного продукта заказчику, а продолжение продуктивных отношений с этим заказчиком. Задачами сопровождения программного продукта являются ликвидация ошибок, которые остались в программном продукте, коррекция проектных спецификаций, улучшение производительности и учет особенностей новой программной и аппаратной среды заказчика, если таковые имеются.

Сопровождение включает следующие виды:

- корректирующее сопровождение (устранение существующих в продукте ошибок без изменения проектных спецификаций);
- обновляющее сопровождение (внесение изменений в спецификации, функциональная коррекция ПО, изготовление нового релиза, улучшающего ПО, с целью при сохранении функциональности увеличения производительности, пропускной способности, количества одновременных пользователей, количества транзакций и т.д.);
- адаптивное сопровождение (адаптация продукта к новой программно-аппаратной среде).

После завершения сопровождения наступает стадия вывода из эксплуатации. Вывод происходит после полного завершения использования ПО. Если функции ПО все еще необходимы, то важно произвести экспорт данных из завершивших работу приложений в новые программные системы. При этом стоимость замены включает стоимость смены технологии — это цена нового ПО, стоимость разработки и поддержки приложений на основе нового программного обеспечения, стоимость затрат персонала на обучение новому ПО и технике работы с ним, а также стоимость краткосрочного падения производительности при замене одних технологий другими.

Рассмотрим подробнее, каким образом осуществляется этап эксплуатации ПО заказчиком, называемый сопровождением. Сопровождение начинается по завершении приемочного тестирования программного продукта, как только все приемочные тесты, которые созданы зачастую с участием или в присутствии закачика, проходят успешно в реальной среде заказчика, т.е. на его программно-аппаратном обеспечении, с реальными данными (как по объему, так и по содержанию), и заказчик удовлетворен результатами. Ес-

тественно, заказчику передается весь программный продукт, т.е. не только код, но и документация, которая включает в себя и сценарии использования, описывающие основную функциональность продукта при разном использовании, и диаграммы классов. Последние описывают основные модули и функции этих модулей в форме методов, взаимодействие между этими классами, а также предметную область, скелетные файлы классов или заготовки сигнатур классов с описанием функциональности этих классов или модулей, взаимодействий с соседними модулями, локальных и глобальных переменных, алгоритмов и структур данных, на основе которых будет работать данный программный продукт, диаграммы последовательности взаимодействия, которые характеризуют как архитектурные особенности программного решения, так и соотношения между различными его составляющими, диаграммы клиент—объект и др. Документация, включающая описание программного продукта с точки зрения пользователя, — это краткое описание основной функциональности, полнофункциональная инструкция пользователя с указанием возможных ошибок, которые могут возникать в работе ПО, описание работы всех его модулей с необходимыми скриншотами, определение терминов, которые могут встречаться в процессе описания программного обеспечения. При этом существует целый ряд видов сопровождения, которые нацелены на решение специфических задач.

Корректирующее сопровождение — необходимый вид. Оно заключается в устранении остаточных сбоев, т.е. существенных ошибок в реализации ПО, которые, несмотря на проведенное тестирование, все еще остаются в продукте и выявляются только расширенным тестированием заказчиком уже в ходе эксплуатации. Естественно, несмотря на то что количество ошибок при тестировании убывает экспоненциально, невозможно устранить все ошибки, и, конечно, большое количество пользователей в различных ситуациях работы одновременно с реальными данными дают возможность выявить достаточно существенное количество весьма серьезных ошибок. Речь идет не о незначительных ошибках, а о таких, которые приводят к остановке системы, критическим сбоям, потере данных, невозможности продолжения работы и т.д. При этом корректирующее сопровождение не включает в себя изменение функциональных требований: речь не идет о переработке продукта с добавлением новой функциональности, речь идет о коррекции.

Другой вид — обновляющее сопровождение, которое как раз нацелено на добавление новой функциональности. Заказчик в ходе эксплуатации ПО достаточно часто приходит к выводу, что некоторые функции, которые не были заложены в изначальных проектных спецификациях, было бы полезно реализовать. Это может происходить по разным причинам: возможно, возникали определенные сложности с бюджетом или сроками реализации, а в процессе эксплуатации возникла потребность добавления новой функциональности (на примере интернет-магазина: не хватает возможности оплаты по кредитным картам, это ведет за собой множество новых требований), что потребует новой итерации разработки, выпуска нового релиза или нового продукта. С точки зрения контракта речь пойдет о дополнительном соглашении к договору, которое предусматривает реализацию этой новой функциональности.

Еще один вид сопровождения — улучшающее, когда заказчик удовлетворен той функциональностью, которая реализована, но возникают дополнительные нефункциональные требования, связанные с тем, что нужно увеличить производительность. На примере интернет-магазина: может возникнуть проблема в пропускной способности интернет-канала из-за того, что количество пользователей существенно превышает запланированное. Ограничениям не удовлетворяет уже и тот сервер БД, который был использован, и та интенсивность транзакций (если они вообще используются) и общая производительность системы, которая для пользователя иногда уже является неудовлетворительной, при этом речь не идет об изменении функциональности.

И еще один вид сопровождения связан с миграцией существующего программного окружения в новую среду. Под программным окружением мы понимаем все множество информационных систем, имеющееся у заказчика, которое как раз и перемещается (например, под управлением новой ОС, нового сервера БД и т.д.). Здесь речь идет уже о том, что необходимо обеспечить интеграцию существующих программных продуктов у заказчика с теми новыми возможностями, которые дают новые программные или аппаратные системы, на которые переходит заказчик. Следует разрабатывать программные продукты таким образом, чтобы они были сопровождаемыми. Сопровождение — это необходимый этап ЖЦ любого проекта, каким бы малым он ни был и какие бы ни были отношения с заказчиком. Почему сопровождение так важно? Во-первых, оно

позволяет выстроить продуктивные, долговременные отношения с заказчиком, поскольку именно этот этап, ради которого собственно и создается ПО, этап промышленной эксплуатации, как правило, является достаточно продолжительно (обычно несколько лет). Именно на этом этапе взаимодействие с заказчиком приносит дополнительные доходы за счет всех вышеописанных вариантов сопровождения. Вторым важным аспектом является то, что сопровождение дает возможность перейти к программному продукту, который можно использоваеть повторно. То есть тот программный продукт, который делался для конкретного заказчика, наверняка, если он хорошо сопровождается, может быть после небольших доработок предложен другому заказчику. Чем правильнее проходит этап сопровождения, тем больше вероятность того, что выпущенный продукт будет устраивать большое количество заказчиков, а в перспективе может быть реализован как коробочный продукт. В этой связи даже для небольших продуктов сопровождение важно.

Следующей стадией является вывод из эксплуатации. Это не самая интересная, может быть, не самая радостная стадия, поскольку ПО верой и правдой служило заказчику достаточно долгое время, к нему уже привыкли, сложились внутренние регламенты, существуют документы, процедуры, пользователям понятно это ПО, они уже достаточно хорошо в нем ориентируются. Но в ряде случаев приходится заказчику сделать вывод о том, что снятие с эксплуатации необходимо. Почему это может быть так? Потому что ПО, в отличие, например, от архитектурных сооружений, морально устаревает достаточно быстро, так как стремительно меняются программные и аппаратные платформы, и в ряде случаев заказчику становится уже невыгодно осуществлять дорогостоящее сопровождение того решения, которое было разработано. Приходится переходить на новое ПО, поддерживающее совершенно новую функциональность, реализация которой слишком дорогостоящая для текущей эксплуатируемой версии. При этом если ряд функций ПО и данные являются еще необходимыми, то важно при миграции обеспечить экспорт данных в новое приложение. Этот процесс непрямолинейный и непростой. Но крайне нежелательно, особенно на уровне КИС с большим количеством важных данных и важностью динамики этих данных, терять эти данные и параметры. Вывод из эксплуатации возможен только на основе взвешенного решения, которое включает оценку стоимости. Стоимость замены

ПО включает целый ряд факторов: стоимость смены технологий (стоимость нового ПО, стоимость лицензий), стоимость разработки и поддержки приложений на основе нового ПО, затраты на обучение персонала, потеря производительности труда персонала в переходный период, поскольку с новым ПО всегда достаточно сложно работать. В ряде случаев вывод из эксплуатации осуществляется вынужденно (например, при обнаружении существенной несовместимости).

Важный проектный документ — план проекта, который включает все стадии ЖЦ: анализ и спецификация требований, первичное и детальное проектирование, реализация, тестирование, интеграция, приемочное тестирование, передача заказчику и сопровождение, вывод из эксплуатации. План проекта также включает основные оценки таких важных измерений проекта, как сроки, стоимость, в том числе в виде общего расписания проекта, которое содержит основные виды деятельности и вехи (milestones, основные границы достижения некоторых результатов). В MSF также важно понятие deliverables — практические результаты, которые получены по достижении каждой вехи. Кроме того, план проекта включает некие более локальные планы: план управления рисками, план тестирования, план интеграции и другие глобальные документы, о которых мы будем говорить дальше.

ЖЦ ПО в зависимости от конкретных его моделей может иметь ряд особенностей. Скажем, проектная спецификация, или написание отдельных компонентов проекта, или особенности архитектуры могут быть не в полной мере детализированы или определены — это зависит от конкретной модели. В ряде моделей существует однопроходный ЖЦ, когда система полностью разрабатывается за один проход по всем стадиям ЖЦ. В других моделях осуществляется итеративное или циклическое повторение этапов ЖЦ с наращиванием или изменением функциональности.

Существует классический подход к разработке ПО на основе структурного анализа и проектирования (structural analysis and development), которое не учитывает ряд аспектов, в частности, в ряде случаев специфику компонентов или модулей кода и выбор языка программирования сложно осуществить до завершения проектных спецификаций (при объектном подходе, например, это можно сделать). Неструктурные аспекты, динамические аспекты проектирования здесь тоже не учитываются. Кроме того, достаточно сложно

осуществить столь масштабное повторное использование кода, как это делается в объектно-ориентированной модели и в подходе, связанном с объектно-ориентированным анализом и проектированием. Нужно сказать, что повторное использование кода — это, пожалуй, важнейшая цель организации ЖЦ ПО. Проблема здесь в том, что ножницы между возможным повторным использованием не только кода, но и других артефактов проекта (документация) составляют до половины стоимости проекта. На этом можно существенно экономить во времени, средствах и людских ресурсах. Но это довольно сложно сделать, так как повторное использование требует хорошей дисциплины проекта, грамотного использования специфических средств. Мы должны к этому стремиться, ряд моделей ЖЦ учитывает это в достаточно большой степени. Кроме того, нужно понимать, что границы фаз ЖЦ могут изменяться и даже перекрываться, в том числе динамически по мере изменения требований в зависимости от модели ЖЦ (например, это имеет место в объектно-ориентированной модели).

В связи с тем что ЖЦ продукта проходит ряд стадий, очевидна необходимость проводить вывод из эксплуатации и переходить к новой версии.

Достаточно интересным является взгляд на вклад различных фаз ЖЦ программных проектов в сроки и стоимость. Анализ произведен на основе целого ряда проектов (порядка 1000), которые велись компанией HP и др. Очевидно, что сопровождение составляет львиную долю стоимости и сроков проекта. При этом такие стадии, как кодирование, даже вместе с тестированием, и анализ требований, даже вместе с изготовлением спецификаций, занимают относительно небольшую долю стоимости. Можно сделать ряд интересных выводов на основе анализа этой динамики. Основные затраты выделяются на сопровождение. Особенно это важно для корпоративных проектов, которые являются долгосрочными и включают большое количество компонентов, которые нужно объединять, интегрировать и поддерживать совместно, что вызывает дополнительные сложности. Кроме того, программные средства, которые увеличивают расширяемость применения ПО, более эффективны, чем все попытки рефакторинга улучшения кода. Еще один интересный вывод состоит в том, что фазы перед кодированием и после него составляют порядка 30% затрат, а собственно кодирование при этом составляет всего 5%. То есть то, что называ-

ется программированием, для больших проектов отнюдь не является затратной частью. Обрамляющие стадии (тестирование и коррекция спецификаций) обеспечивают существенное улучшение качества ПО и его соответствие требованиям. Нужно сказать, что правильная постановка обрамляющих стадий очень важна и ускоряет кодирование.

Большая часть серьезных ошибок, которые выявляются в программных проектах, происходит на стадиях проектирования и построения спецификаций. Поэтому эти ошибки очень дорогостоящи, поскольку приходится переделывать и код, и документацию, и нужны формальные методы их анализа (это и ревизия проекта, и более формальные логические технологии проверки корректности). Цена ошибки растет примерно экспоненциально по мере продвижения проекта по жизненному циклу. Если ошибка обнаруживается на стадии анализа требований, то ее исправление достаточно дешево. Если же она обнаружена на более поздних стадиях, особенно на стадии сопровождения, то ее цена во много раз выше, потому что приходится изменять всю версию ПО, документацию, диаграммы и целый ряд других программных продуктов. К счастью, есть специальные средства для поиска, выявления и устранения ошибок.

Каждая фаза ЖЦ ПО включает три важных составляющих — процессы, методы и средства. Под процессами понимают задачи, которые необходимо реализовать, они отличаются и, по сути, не зависят друг от друга. Методы — это относительно формальное описание каждой задачи процесса. Средства — автоматический инструментарий типа CASE-средств для поддержки процессов и методов.

Ниже перечислены основные виды моделей ЖЦ, которые будут подробнее рассмотрены в следующих главах: это прежде всего модель Build-and-fix — «кодируй и фиксируй», по сути, она близка к модели проб и ошибок; затем — водопадная модель, которая дает возможность за один проход ЖЦ построить полномасштабное программное обеспечение; модель быстрого прототипирования, которая, как правило, объединяется с другими моделями; инкрементальная модель с последовательным наращиванием функциональности; модель синхронизации и стабилизации, или модель Microsoft; спиральная модель, в которой очень важна оценка рисков и которая тоже подразумевает циклическую обработку продукта по мере его движения к пользователю; объектно-ориентированная модель с перекрытием ряда фаз и во многом циклическим или итерационным развитием.

Какие общие черты можно выделить в перечисленных моделях ЖЦ? Как правило, они включают все его стадии, за исключением модели Build-and-fix. Кроме того, предполагается несколько итераций по разработке продукта, за исключением каскадной модели. Как правило, стадии ЖЦ четко различимы, кроме объектно-ориентированной модели, где они могут объединяться. Некоторые отдельные модели, связанные с некоторыми методологиями проектирования, такие как модель синхронизации и стабилизации, связаны с методологией MSF. RUP поддерживает каскадные и спиральные сценарии жизненного цикла и т.д.

Грамотное применение модели ЖЦ требует высокой организационной зрелости команды и серьезной дисциплины проекта с точки зрения стандартов документирования, кодирования, использования специализированных CASE-средств и т.д. Если такие знания недостаточны, то объектно-ориентированная модель может выродиться в такую модель, как Build-and-fix, т.е. можно потерять все преимущества модели и увеличить затраты.

Таким образом, универсальной модели ЖЦ не существует, все определяется характером и масштабом проекта. Каждая модель имеет свои преимущества и свои недостатки. Об этом мы поговорим подробнее в следующей главе.

Что определяют модели ЖЦ программных продуктов? Во-первых, характер и масштаб проекта. В этой связи, как только при анализе и спецификации требований и ограничений определены основные границы проекта и продукта, в идеале следует определиться с совокупностью моделей, которые будут выбраны. Здесь критичны объем продукта, сроки и проектные риски. Скажем, спиральная модель существенно связана с использованием рисков, поэтому ее имеет смысл применять в случаях, когда необходим анализ рисков. Модели определяют экономику проекта, в том числе и скорость возврата инвестиций. В случае если нет острой необходимости применять модель полного ЖЦ, можно сэкономить на отдельных стадиях, например если не нужно разрабатывать документацию в полном объеме. Модель определяет степень сопровождаемости: в ряде моделей мы можем получить продукт, который будет более сопровождаемым. Модели ЖЦ определяют также перспективы развития — насколько можно будет удовлетворить будущие запросы клиента. Также модели ЖЦ определяют общую структуру проекта с точки зрения его эволюционного или революцион-

ного совершенствования: потребуются ли радикальные изменения или можно ограничиться архитектурой, в которой проект будет стабильно эволюционировать. Модели определяют скорость поиска и устранения ошибок, например, модель синхронизации и стабилизации нацелена на частое раннее тестирование. Некоторые модели, как уже отмечалось, способствуют хорошему управлению рисками проекта. Кроме того, отдельные модели подразумевают изготовление прототипов (модель быстрого прототипирования, инкрементальная модель), другие требуют изготовления готового продукта сразу же.

Какие особенности ЖЦ можно выделить уже в первом приближении для конкретных моделей? Модель Build-and-fix — это модель неполного ЖЦ, которая пригодна для малых проектов (≈1000 строк) и абсолютно непригодна для больших и сложных проектов с большим потенциалом развития. Водопадная, или каскадная, модель обеспечивает хорошую обратную связь с ранними стадиями ЖЦ, поскольку завершается подготовкой документов, которые позволяют перейти к следующей стадии. Без этих документов, без корректного закрытия предыдущей стадии невозможно начало следующей. Быстрое прототипирование — несамостоятельная модель, не приводящая к созданию надежного кода. Инкрементальная модель всегда дает возможность получить на каждом этапе готовый продукт, пусть и неполнофункциональный. Модель синхронизации и стабилизации, или модель Microsoft, нацелена на раннее выявление ошибок. Спиральная модель подразумевает несколько итераций и нацелена на анализ рисков. Объектно-ориентированная модель — это итеративное проектирование с перекрытием фаз и наложением их друг на друга.

Уже говорилось о том, что цена поиска ошибок экспоненциально растет по мере продвижения к завершению, поэтому ошибки нужно обнаруживать как можно раньше. Для этого существуют специальные методы, которые содержатся на всех стадиях жизненного цикла и включают процессы, методы и средства.

Кратко остановимся на преимуществах и недостатках различных моделей ЖЦ.

Модель Build-and-fix хороша для небольших проектов, которые не требуют сопровождения, но абсолютно непригодна для корпоративных или вообще нетривиальных проектов объемом более 1000 строк.

Водопадная модель является документно-управляемой, поскольку документы фиксируют завершение каждой стадии и обеспечивают четкую дисциплину проекту. Но в итоге, поскольку это однопроходная модель, ПО может не соответствовать требованиям клиента.

Модель быстрого прототипирования вызывает соблазн повторного использования кода, который не является достаточно протестированным, хорошо задокументированным и который, вообще говоря, следует заново реализовать как ненадежный. Но эта модель позволяет выявить соответствие ПО требованиям клиента, т.е. обеспечить анализ требований и выявление наиболее важных для клиента.

Инкрементальная модель способствует хорошей сопровождаемости за счет того, что получается достаточно плавный путь перехода от одной версии к другой. Эта модель способствует раннему возврату инвестиций, но требует открытой архитектуры, которая поддерживает такое эволюционное совершенствование программного продукта, и может выродиться в модель Build-and-fix.

Модель синхронизации и стабилизации удовлетворяет будущим потребностям клиента и обеспечивает высокую интеграцию компонентов, но достаточно сложна, поскольку требует интенсивного тестирования. Поэтому она не получила широкого применения вне Microsoft.

Спиральная модель объединяет характеристики перечисленных выше моделей, но желательно использовать ее во внутренних проектах, поскольку она требует тщательного анализа рисков, и ряд допущений, связанных с рисками, может не быть передан внешнему разработчику.

Объектно-ориентированная модель требует дисциплины, может выродиться в модель проб и ошибок и обеспечивает итеративную разработку и параллелизм взаимодействия между фазами.

На что влияет выбор модели ЖЦ? На скорость разработки, время выхода проекта на рынок, качество и стоимость продукта, стратегию управления изменениями и рисками, отношения с заказчиком на стадии сопровождения.

Окончательные выводы, которые можно сделать по моделям ЖЦ: выбор модели определяет основные критические параметры проекта — это успех проекта в целом, архитектура проекта, его бюджет, в каких случаях можно сэкономить. Модель должна быть адекватна опыту проектной команды с точки зрения знаний пред-

метной области и знания конкретных технологий, CASE-средств, документирования, подходов к документированию и т.д. Серьезные модели, такие как спиральная или объектно-ориентированная, требуют определенной дисциплины и зрелости. В противном случае они вырождаются в модель проб и ошибок. Универсальной модели не существует. Выбор модели определяется исключительно характером и масштабом проекта. Ряд моделей можно комбинировать. У каждой модели есть свои преимущества и недостатки, которые обнаруживаются и имеют смысл только в контексте проекта, с учетом его особенностей.

Еще несколько слов о том, что помогает в программной инженерии, в изготовлении корпоративных решений. Это CASE-средства и CASE-технологии. ПО имеют целый ряд аспектов. ПО в малом можно рассматривать как искусство программирования или разработку отдельных модулей, отдельных фрагментов кода. ПО в большом можно понимать как software engineering, это технологии программирования, обеспечение жизненного цикла ПО с теми этапами и теми моделями, о которых было сказано выше. И еще один аспект — это командная работа ПО в массе, поддержка коллективной разработки, что очень важно для корпоративных информационных систем, в разработке которых участвуют целые коллективы разработчиков и тратят массу времени на взаимодействие, интеграцию, совместную разработку, командную работу.

Одним из важных CASE-средств, которое мы будем рассматривать, является Visual Studio.NET от Microsoft. О нем мы будем говорить в дальнейшем. Существует большое количество других CASE-средств: линейка Rational, которая поддерживает RUP. CASE-средства помогают во всех трех аспектах — и узко при кодировании, и при оптимизации ЖЦ, и при командной работе.

CASE-средства в первом приближении делятся на CASE-средства верхнего уровня (front-end), т.е. соответствуют первичным стадиям ЖЦ, и нижнего уровня (back-end), соответствующие стадиям ЖЦ, начиная с реализации. Важно отметить, что существуют конвейерные средства, такие как линейка Rational, Microsoft Visual Studio.NET, которые представляют собой среды, т.е. наборы определенного инструментария или своего рода конвейеры для выполнения связанных операций компиляции, тестирования, интеграции, редактирования кода, изготовления проектной документации, диаграммирования и т.д.

CASE-технологии дают неоспоримое преимущество при изготовлении больших программных систем. Но при своем применении они требуют определенных условий, таких как организационная зрелость команды, знание стандартов (UML, XML), знание самого средства. Кроме того, CASE-средства применимы для больших проектов корпоративных систем. Для небольших проектов стоимость CASE-средств и обучения им неоправданно высока. В результате успешного применения CASE-средств можно получить существенный рост производительности труда разработчиков и, в результате, если мы говорим о проекте в целом, существенное снижение сроков и стоимости программного проекта.

Какие метрики ПО применяются при контроле за ЖЦ программного проекта? Для проекта в целом это сроки, стоимость и функциональность — так называемый проектный треугольник компромиссов. В ряде случаев имеет смысл проводить анализ cost-benefit, т.е. анализ тех преимуществ, которые получает заказчик в зависимости от тех или иных вложений. Таким образом, этот треугольник имеет смысл рассматривать во взаимосвязи его основных характеристик и параметров. Наконец, для конкретных стадий ЖЦ (скажем, тестирования и сопровождения) можно выделить специфические метрики. Вообще говоря, для каждого этапа они свои. В случае тестирования можно использовать такие метрики, как сложность отдельного модуля, количество строк (обычно это тысячи строк), количество различных операторов или операндов, которые используются в том или ином модуле или фрагменте кода, относительное количество ошибок, которые выявлены на 1000 строк кода. Для стадии сопровождения это отслеживание и исправление допущенных ранее ошибок, поскольку не все ошибки проекта могут быть выявлены непосредственно на стадии реализации и до передачи заказчику. Нужно анализировать общее количество сбоев, коммуникацию или взаимодействие по ним. Здесь работают такие метрики, как состояние сбоев и отчетов. Кроме того, выявление источника и определенное состояние дискуссии, результаты (удалось устранить этот сбой, насколько он серьезный), а также метрики предыдущих стадий. Важные выводы, которые можно сделать, сводятся к тому, что решение принимается менеджером проекта: стоит ли прекратить тестирование, передать в эксплуатацию или нет? И, как правило, простые метрики являются достаточным.