# Методологии разработки программного обеспечения

## Введение

Вначале сформулируем цели, без которых невозможен ни один проект. Основная задача любого успешного проекта заключается в том, чтобы на момент запуска системы и в течение всего срока ее эксплуатации можно было обеспечить:

• требуемую функциональность системы и степень адаптации к изменяющимся условиям ее функционирования;

• требуемую пропускную способность системы;

• требуемое время реакции системы на запрос;

• безотказную работу системы в требуемом режиме, то есть: готовность и доступность системы для обработки запросов пользователей;

• простоту эксплуатации и поддержки системы;

• требуемую безопасность.

Производительность является главным фактором, который определяет эффективность системы. Хорошее проектное решение — основа высокопроизводительной системы.

Проектирование информационных систем охватывает три основные области:

• проектирование объектов данных, которые будут реализованы в базе данных;

• проектирование программ, экранных форм, отчетов, которые будут обеспечивать выполнение запросов к данным;

• учет конкретной среды или технологии: топологии сети, конфигурации аппаратных средств, использования архитектур «файл-сервер», «клиент-сервер», параллельной обработки, распределенной обработки данных и т.п.

В реальных условиях проектирование информационных систем — это поиск способа, который обеспечивает необходимую функциональность системы средствами имеющихся технологий с учетом заданных ограничений.

Под методологией разработки подразумевается набор методов и критериев оценки, которые используются для постановки задачи, планирования, контроля и в конечном итоге — для достижения поставленной цели. Сам процесс разработки описывается моделью, которая определяет последовательность наиболее общих этапов и получаемых результатов.

Долгое время процесс разработки ПО осуществлялся в соответствии с методиками, наработанными в инженерной области, — стандартная практика поэтапного создания продукта, начиная с составления спецификаций и заканчивая поставкой заказчику. Существуют стандарты ГОСТ (Россия) и ISO (Европа, Россия), CMM (Capability Maturity Model — распространен в США), регламентирующие данный процесс.

Известны несколько основных моделей жизненного цикла ПО.

Каскадная модель — переход на следующий этап означает полное завершение работ на предыдущем этапе.

Поэтапная модель с промежуточным контролем — разработка ПО ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют уменьшить трудоемкость процесса разработки по сравнению с каскадной моделью, но время жизни каждого из этапов растягивается на весь период разработки.

Спиральная модель — особое внимание уделяется начальным этапам разработки: выработке стратегии, анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования). Каждый виток спирали предполагает создание некой версии продукта или какого-либо его компонента; при этом уточняются характеристики и цели проекта, определяется его качество и планируются работы следующего витка спирали.

Активное программирование и его клоны — наиболее популярным для данной модели стало экстремальное программирование (extreme Programming, XP). Отцом-идеологом XP считают Кента Бека (Kent Beck). XP является довольно молодой методологией, оценки которой весьма противоречивы: от восторженных до резко негативных. Основными принципами являются простота решений и интенсивная разработка малыми группами, активное общение в группе и обратная связь с клиентом, фактически вовлеченным в процесс разработки, а также известная доля куража.

Ниже мы рассмотрим классические модели, а об экстремальном программировании расскажем в отдельной части данной статьи.

Жизненный цикл программного обеспечения (ПО) представляет собой модель его создания и использования. Данная модель отражает его различные состояния, начиная с момента возникновения потребности в данном ПО и заканчивая моментом его полного выхода из употребления у всех пользователей. Модели различаются способом взаимосвязи этапов жизненного цикла, но каждый из этих этапов в том или ином виде присутствует в каждой модели. Ниже мы последовательно рассмотрим все эти этапы.

### Стратегия

Определение стратегии предполагает обследование системы. Основная задача обследования — это оценка реального объема проекта, его целей и задач, а также получение определений сущностей и функций на высоком уровне. На этом этапе привлекаются высококвалифицированные бизнес-аналитики, которые имеют постоянный доступ к руководству фирмы; также предполагается тесное взаимодействие с основными пользователями системы и бизнес-экспертами. Основная задача такого взаимодействия — получить как можно более полную информацию о системе, однозначно понять требования заказчика и передать данную информацию в формализованном виде системным аналитикам. Как правило, информация о системе может быть получена на основании ряда бесед (или семинаров) с руководством, экспертами и пользователями.

Итогом этапа определения стратегии становится документ, где четко сформулировано следующее: что именно причитается заказчику, если он согласится финансировать проект; когда он сможет получить готовый продукт (график выполнения работ); во сколько это ему обойдется (график финансирования этапов работ для крупных проектов). В документе должны быть отражены не только затраты, но и выгода, например время окупаемости проекта, ожидаемый экономический эффект (если его удается оценить).

Определение стратегии — это принципиальный ответ на вопросы типа: «Будем ли мы делать этот проект за такие-то деньги или нет?» или «Делаем ли мы в принципе этот проект с данным разработчиком?» Иными словами, в результате определения стратегии разработчик и заказчик принимают решение о продолжении работ на определенных условиях с определенными обязанностями сторон.

Следует выделить набор фактов, которые должны быть обязательно отражены в заключительном документе после проведения обследования и анализа деятельности предприятия:

• ограничения, риски, критические факторы, влияющие на проект;

• совокупность условий эксплуатации будущей системы: архитектура, аппаратные и программные ресурсы, внешние условия ее функционирования; состав исполнителей и работ, обеспечивающих функционирование системы;

• критические сроки завершения этапов, форма сдачи работ, защита коммерческой информации;

• описание выполняемых системой функций;

• интерфейсы и распределение функций между человеком и системой;

• требования к программным и информационным компонентам ПО;

• наличие потенциального развития системы в будущем;

• то, что не будет реализовано в рамках проекта.

Данный этап жизненного цикла ПО может быть представлен в модели только один раз, особенно если модель имеет циклическую структуру, однако это не означает, что в циклических моделях стратегическое планирование производится раз и навсегда. Циклические модели предназначены для решения проблем ПО, которое развивается по времени; анализ текущего состояния ПО и предприятия, его использующего, производится на этапе анализа. Таким образом, в циклических моделях этапы определения стратегии и анализа как бы склеиваются, а вероятность их разделения существует лишь на самом первом витке, когда руководство предприятия принимает принципиальное решение о старте проекта. В целом этап анализа посвящен разработке документа уровня руководства предприятия.

### Анализ

Этап анализа предполагает подробное исследование бизнес-процессов (функций, определенных на предыдущем этапе) и информации, необходимой для их выполнения (сущностей, их атрибутов и связей (отношений)). Этот этап дает информационную модель, а следующий за ним этап проектирования — модель данных.

Вся информация о системе, собранная на этапе определения стратегии, формализуется и уточняется на этапе анализа. Особое внимание следует уделить полноте переданной информации, анализу информации на непротиворечивость, а также поиску неиспользуемой или дублирующейся информации. Как правило, заказчик вначале формирует требования не к системе в целом, а к отдельным ее компонентам. И в этом конкретном случае циклические модели жизненного цикла ПО имеют преимущество, поскольку с течением времени с большой вероятностью потребуется повторный анализ, так как у заказчика зачастую аппетит приходит во время еды. На этом же этапе определяются необходимые компоненты плана тестирования.

Аналитики собирают и фиксируют информацию в двух взаимосвязанных формах:

• функции — информация о событиях и процессах, которые происходят в бизнесе;

• сущности — информация о вещах, которые имеют значение для организации и о которых что-либо известно.

Ранее двумя классическими результатами анализа были: иерархия функций, которая разбивает процесс обработки на компоненты («что делается и из чего это состоит»), и модель «сущность-связь» (Entry Relationship model, ER-модель), которая описывает сущности, их атрибуты и связи (отношения) между ними. Эти результаты являются необходимыми, но не достаточными. К достаточным результатам следует отнести диаграммы потоков данных и диаграммы жизненных циклов сущностей, которые описывают динамику системы. В настоящее время существует способ формализации проекта — Unified Modelling Language (UML), позволяющий формально описать различные стороны жизнедеятельности разрабатываемого проекта. Существует достаточно большое количество ПО, реализующего UML, например Rational Rose, Microsoft Visio. О UML мы расскажем в отдельной части данной статьи.

### Проектирование

На этапе проектирования формируется модель данных. Проектировщики получают входные данные анализа. Конечным продуктом этапа проектирования являются схема базы данных (если таковая существует в проекте) или схема хранилища данных (ER-модель) и набор спецификаций модулей системы (модель функций).

В случае небольшого проекта одни и те же люди могут выступать в роли и аналитиков, и проектировщиков, и разработчиков. Возникает вопрос, насколько актуальна передача результатов самому себе. В принципе, достаточно актуальна, поскольку часто помогает найти, например, не описанные вообще, нечетко описанные, противоречиво описанные компоненты системы и прочие недостатки, способствует предотвращению потенциальных ошибок, а также даст возможность еще раз подумать.

Все спецификации должны быть предельно точными. План тестирования системы также дорабатывается на этом этапе разработки. Во многих проектах результаты этапа проектирования оформляются в виде единого документа — так называемой технической спецификации. Здесь стоит упомянуть о преимуществах способа UML, который позволяет получить одновременно как документы анализа, которые отличаются меньшей детализацией (их потребители — менеджеры производства), так и документы проектирования (их потребители — менеджеры групп разработки и тестирования). Кроме того, ПО, реализующее UML, позволяет осуществить генерацию кода — как минимум иерархию классов, а также некоторые части кода самих методов.

Задачами проектирования являются:

• рассмотрение результатов анализа и проверка их полноты;

• семинары с заказчиком;

• определение критических участков проекта и оценка ограничений проекта;

• определение архитектуры системы;

• принятие решения об использовании продуктов сторонних разработчиков, а также о способах интеграции и механизмах обмена информации с этими продуктами;

• проектирование хранилища данных: модель базы данных, бета-версия базы данных;

• проектирование процессов и кода: окончательный выбор средств разработки, определение интерфейсов программ, отображение функций системы на ее модули и определение спецификаций модулей;

• определение требований к процессу тестирования;

• определение требований безопасности системы.

### Реализация

При реализации проекта важно координировать группу (группы) разработчиков. Все разработчики должны подчиняться жестким правилам контроля исходных тестов. Группа разработчиков, получив технический проект, начинает писать код модулей. Основная их задача состоит в том, чтобы уяснить спецификацию: проектировщик написал, что надо сделать, разработчик определяет, как это сделать.

На этапе разработки осуществляется тесное взаимодействие проектировщиков, разработчиков и групп тестировщиков. В случае интенсивной разработки тестировщик буквально неразлучен с разработчиком, фактически становясь членом группы разработки.

Проектировщик на этапе разработки выполняет функцию «ходячего справочника», поскольку должен постоянно отвечать на вопросы разработчиков, касающиеся технической спецификации.

Чаще всего на этапе разработки меняются интерфейсы пользователя. Это обусловлено, наряду с прочим, периодической демонстрацией модулей заказчику. Также могут существенно изменяться запросы к данным.

Следует отметить необходимость существования выделенного рабочего места, где происходит сборка всего проекта. Именно эти модули передаются на тестирование. Взаимодействие тестировщика и разработчика без централизованной передачи допустимо только в том случае, если срочно требуется проверить какую-то правку. Очень часто этап разработки сопряжен с этапом тестирования, и оба процесса идут параллельно. Синхронизирует действия тестеров и разработчиков система bug tracking.

Кроме того, должны быть организованы хранилища готовых модулей проекта и библиотек, которые используются при сборке модулей. Это хранилище постоянно обновляется. Контролировать процесс обновления должен один человек. Одно хранилище создается для модулей, прошедших функциональное тестирование, второе — для модулей, прошедших тестирование связей. Первое — это черновики, второе — то, из чего уже можно собирать дистрибутив системы и демонстрировать его заказчику для проведения контрольных испытаний или для сдачи каких-либо этапов работ.

Следует отметить исключительную важность обмена информацией между проектировщиками, разработчиками, тестировщиками: ошибки должны быть классифицированы согласно приоритетам; для каждого класса ошибок должна быть определена четкая структура действий: «что делать», «как срочно», «кто ответственен за результат»; каждая проблема должна отслеживаться проектировщиком/разработчиком/тестировщиком, отвечающим за ее устранение. То же самое касается ситуаций, когда нарушаются запланированные сроки разработки, передачи модулей на тестирование. Все проблемы при взаимодействии между группами могут стоить достаточно дорого.

### Тестирование

Группы тестирования могут привлекаться к сотрудничеству уже на ранних стадиях разработки проекта. Строго говоря, комплексное тестирование следует выделить в отдельный этап разработки. В зависимости от сложности проекта тестирование и исправление ошибок может занимать треть, половину общего времени работы над проектом и даже больше.

Чем сложнее проект, тем больше будет потребность в автоматизации системы хранения ошибок — bug tracking, которая обеспечивает следующие функции:

• хранение сообщения об ошибке (к какому компоненту системы относится ошибка, кто ее нашел, как ее воспроизвести, кто отвечает за ее исправление, когда она должна быть исправлена);

• система уведомления о появлении новых ошибок, об изменении статуса известных в системе ошибок (уведомления по электронной почте);

• отчеты об актуальных ошибках по компонентам системы;

• информация об ошибке и ее история;

• правила доступа к ошибкам тех или иных категорий;

• интерфейс ограниченного доступа к системе bug tracking для конечного пользователя.

Подобные системы берут на себя множество организационных проблем, в частности вопросы автоматического уведомления об ошибках.

Собственно тесты систем можно разделить на несколько категорий:

• автономные тесты модулей; они используются уже на этапе разработки компонентов системы и позволяют отслеживать ошибки отдельных компонентов;

• тесты связей компонентов системы; эти тесты также используются и на этапе разработки, и на этапе тестирования, они позволяют отслеживать правильность взаимодействия и обмена информацией компонентов системы;

• системный тест; он является основным критерием приемки системы; как правило, это группа тестов, включающая и автономные тесты, и тесты связей и модели; данный тест должен воспроизводить работу всех компонентов и функций системы; основная цель данного теста — внутренняя приемка системы и оценка ее качества;

• приемосдаточный тест; основное его назначение — сдать систему заказчику; здесь разработчики часто занижают требования к системе по сравнению с системным тестом, и причины этого вполне очевидны;

• тесты производительности и нагрузки; данная группа тестов входит в системный тест, но достойна отдельного упоминания, поскольку именно эта группа тестов является основной для оценки надежности системы.

В тесты каждой группы обязательно входят тесты моделирования отказов. Здесь проверяется реакция компонента, группы компонентов, системы в целом на отказы вида:

• отказ отдельного компонента информационной системы;

• отказ группы компонентов информационной системы;

• отказ основных модулей информационной системы;

• отказ операционной системы;

• жесткий сбой (отказ питания, жестких дисков).

Эти тесты позволяют оценить качество подсистемы восстановления корректного состояния информационной системы и служат основным источником информации для разработки стратегий предотвращения негативных последствий сбоев при промышленной эксплуатации. Как правило, этим классом тестов разработчики традиционно пренебрегают, а затем вынуждены «лечить» последствия сбоев на промышленной системе.

Еще одним важным аспектом программы тестирования информационных систем является наличие генераторов тестовых данных. Они используются для проведения тестов функциональности системы, тестов надежности системы и тестов производительности системы. Задачу оценки характеристик зависимости производительности информационной системы от роста объемов обрабатываемой информации без генераторов данных решить невозможно.

### Внедрение

Опытная эксплуатация перекрывает процесс тестирования. Система редко вводится полностью. Как правило, это процесс постепенный или итерационный — в случае циклического жизненного цикла.

Ввод в эксплуатацию проходит как минимум три стадии:

• первоначальная загрузка информации;

• накопление информации;

• выход на проектную мощность (то есть собственно переход к этапу эксплуатации).

Первоначальная загрузка информации инициирует довольно узкий спектр ошибок: в основном речь идет о проблемах рассогласования данных при загрузке и о собственных ошибках загрузчиков. Здесь требуется применить методы контроля качества данных (в противном случае в дальнейшем наведенные ошибки обойдутся намного дороже). Это то, чего не было или не могло быть отслежено на тестовых данных. Такие ошибки должны быть исправлены как можно быстрее. Не поленитесь поставить отладочную версию системы (если, конечно, вам позволят развернуть весь комплекс сопровождающего отладку информационной системы ПО на месте). Если отладку «на живых» данных произвести невозможно, вам придется моделировать ситуацию, и как можно быстрее. В этом случае требуются очень квалифицированные тестировщики.

В период накопления информации из информационной системы выявляется наибольшее количество ошибок, связанных с многопользовательским доступом. Часто на этапе тестирования им не уделяется должного внимания — из-за сложности моделирования и дороговизны средств автоматизации процесса. Вторая категория исправлений связана с тем, что пользователя не устраивает интерфейс. Здесь не всегда нужно выполнять абсолютно все пожелания пользователя, иначе процесс ввода в эксплуатацию будет бесконечным. В данный момент циклические модели и модели с обратной связью этапов также позволяют снизить затраты.

Этот этап является также наиболее серьезным тестом — тестом одобрения пользователем (customer acceptance tests). Если этого не было сделано на этапе тестирования, то на этапе внедрения это непременно произойдет, и вашим тестировщиком фактически будет ваш собственный заказчик, что не всегда приемлемо, особенно если для последнего это будет несколько неожиданно.

Выход системы на проектную мощность в хорошем варианте — это доводка мелких ошибок и редкие серьезные ошибки.

### Эксплуатация и техническая поддержка

Здесь последним документом, от которого зависят разработчики, является документ технической приемки. Итак, проект сдан, разработка завершена, ошибки, с которыми заказчик согласился, описаны в документации, и в идеале стороны довольны друг другом. Документ также определяет необходимый персонал и требуемое оборудование для поддержки работоспособности системы, а также условия нарушения эксплуатации продукта и ответственность сторон. Помимо этого, если документ заключается между сторонами, он содержит условия технической поддержки.

Условия технической поддержки и ситуации, которые подпадают под понятие «техническая поддержка», а также условия оплаты определяются, как правило, в отдельном документе. Иногда в рекламных целях менеджеры разработки пытаются включить в документы обязательства гарантии исправления ошибок на условиях технической поддержки за «икс» дней. Такой подход, возможно, и хорош с рекламной точки зрения, но следует отдавать себе отчет в том, что невыполнение обязательств по техническим причинам намного хуже, чем отсутствие «вкусной» рекламы.

## Каскадная модель

Данная модель также носит название «водопад». Классическими представителями реализации данной методологии являются стандарты ISO и CMM.

Модель предполагает следующие свойства взаимодействия этапов:

• модель состоит из последовательно расположенных этапов;

• каждый этап полностью заканчивается до того, как начнется следующий;

• этапы не перекрываются во времени: следующий этап не начинается до тех пор, пока не завершится предыдущий;

• возврат к предыдущим этапам не предусмотрен либо всячески ограничен;

• исправление ошибок происходит лишь на стадии тестирования;

• результат появляется только в конце разработки.

Критерием появления результата является отсутствие ошибок и точное соответствие продукта первоначальной спецификации.

## Поэтапная модель с промежуточным контролем

Данная модель еще известна как итерационная модель или «водоворот».

Модель характеризуется следующими свойствами взаимодействия этапов:

• модель состоит из последовательно расположенных этапов (точно так же, как и «водопад»);

• каждый этап имеет обратную связь с предыдущими этапами;

• исправление ошибок происходит на каждом из этапов, сразу при выявлении проблемы — это промежуточный контроль;

• этапы перекрываются во времени по причине наличия обратной связи: следующий этап не начинается, пока не завершится предыдущий; при первом проходе по модели вниз, как только обнаружена ошибка, осуществляется возврат снизу вверх к предыдущим этапам, которые повлекли ошибку; таким образом, фактически этапы оказываются растянутыми во времени;

• результат появляется только в конце разработки, как и в модели «водопад».

Критерием появления результата является приемлемое качество продукта, то есть такое состояние продукта, когда наиболее критические для клиента ошибки устранены, а с наличием непринципиальных для жизнедеятельности системы ошибок клиент согласилcя — данные ошибки описаны в документации и фактически переведены таким образом в разряд особенностей системы.

## Спиральная модель

Коммерческими представителями данной методологии являются RUP (Rational Unified Process), MSF (Microsoft Consulting Services), и об этом будет рассказано в следующих частях данной публикации.

Результат появляется фактически на каждом витке спирали. Этот результат, который является промежуточным, анализируется, а затем выявленные недостатки продукта становятся поводом для инициирования следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в итоге выбирается обоснованный вариант, который доводится до реализации. Спираль завершается тогда, когда клиент и разработчик приходят к согласию относительно результата.

Модель предполагает также свойства взаимодействия этапов:

• модель состоит из последовательно расположенных этапов (как и «водопад») в пределах одного витка спирали;

• внутри витка спирали этапы не имеют обратной связи; анализ результата осуществляется в конце витка и инициирует новый виток спирали;

• исправление ошибок происходит на этапе тестирования на каждом из витков спирали; фактически часть ошибок исправляется в пределах одного витка посредством связи этапов кодирования и тестирования; ошибки, которые не могут быть исправлены и требуют более глубоких структурных изменений, инициируют новый виток спирали;

• этапы могут перекрываться во времени в пределах одного витка спирали;

• результат появляется в конце каждого витка спирали и подвергается подробному анализу, анализируются новые требования заказчика и инициируется новый виток спирали;

• при переходе от витка к витку происходит накопление и повторное использование программных средств, моделей и прототипов;

• процесс ориентирован на развитие и модификацию системы в процессе ее проектирования, на анализ рисков и издержек в процессе проектирования.

Отметим, что основная особенность данной методологии состоит в концентрации сложности на начальных этапах жизненного цикла ПО (анализ, проектирование); при этом сложность и трудоемкость последующих этапов в пределах одного витка спирали относительно невысокие. По этой методологии предлагается способ снижения затрат в целом при разработке ПО за счет предотвращения потенциальных ошибок на этапах анализа и проектирования. Этап определения стратегии присутствует на первом витке спирали либо «склеен» с этапом анализа первого витка спирали.