Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

М. Ю. Катаев

# ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

Учебное пособие

Томск
«Эль Контент»
2013

УДК 004.41(075.8)
ББК 32.973.26-018.2я73
К 290

Рецензенты:
**Гусев Е. В.**, канд. геол.-минерал. наук, доцент Томского политехнического университета;
**Козин Е. С.**, ведущий программист Института химии нефти Сибирского отделения Российской Академии наук.

**Катаев М. Ю.**

К 290 Введение в программную инженерию : учебное пособие / М. Ю. Катаев. — Томск : Эль Контент, 2013. — 160 с.

ISBN 978-5-4332-0062-3

Учебное пособие содержит материалы, отражающие методологические основы современной программной инженерии, обеспечивающей жизненный цикл (ЖЦ) разрабатываемых программных средств (ПС). Представлены профили международных стандартов ЖЦ систем и комплексов программ, регламентирующие в программной инженерии модели и процессы управления проектами ПС. Рассмотрены процессы управления проектами, формированию состава группы, возможные риски в ЖЦ ПС. Представлены стандартизированные характеристики качества программных средств. Учебное пособие целесообразно использовать при обучении студентов как вводный курс в направление «Программная инженерия».

УДК 004.41(075.8)
ББК 32.973.26-018.2я73

ISBN 978-5-4332-0062-3

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

Информационные технологии (ИТ) в настоящее время являются двигателем экономического прогресса во всем мире. Важнейшей составляющей ИТ, в которую вложены огромные средства и знания нескольких поколений ученых, является программное обеспечение информационных систем. Благодаря своим особенностям программирование, которое сейчас стало направлением индустрии, оказалось в центре глобализации мировой экономики. По данным исследования всемирной организации Business Software Alliance (BSA), опубликованного 8 декабря 2005 года, мировая ИТ-индустрия, в которой работает более 1 миллиона предприятий, вносит в мировую экономику почти 2 триллиона долларов.

В России наблюдается устойчивый рост рынка ИТ-услуг начиная с конца 90-х годов, когда по данным отчёта Vendor Shares в 2004 г. он составил 26,3%, что соответствует $1,9 млрд. Средний ежегодный прирост российского рынка ИТ-услуг в прогнозируемый пятилетний период составит 25,4%, а его объём к 2009 г. достигнет $5,8 млрд. По темпам роста он значительно опережает рынки ИТ-услуг в странах как Восточной, так и Западной Европы.

Программные системы ныне присутствуют повсеместно: практически любые электронные устройства содержат программное обеспечение того или иного вида. Без соответствующего программного обеспечения в современном мире невозможно представить индустриальное производство, школы и университеты, систему здравоохранения, финансовые и правительственные учреждения. Многие используют программное обеспечение для самообразования или различного рода развлечений. В таких системах стоимость ПО порой составляет большую часть общей стоимости продукта. Более того, стоимостные показатели отраслей, занимающихся производством ПО, становятся определяющими для экономики — как национальной, так и международной (особенно это относится к производству и использованию суперкомпьютеров) [1].

Активным потребителем продукции и услуг в сфере ИТ в Российской Федерации является государство. Как и в развитых странах, доля спроса со стороны государства в течение последних 5 лет на российском рынке ИТ достигала 30 процентов, что стало существенным стимулом роста отрасли. Значительный объем спроса приходится на несколько крупных компаний, находящихся под контролем государства (ОАО «Газпром», РАО «Российские железные дороги», РАО «ЕЭС России», ОАО «Аэрофлот», ОАО «Связьинвест»). Активным источником спроса на

рынке ИТ являются также предприятия финансовой и нефтегазовой отрасли, связи и торговли, оборонной промышленности.

Российский ИТ-рынок в значительной мере тяготеет к крупным проектам. Традиционно, крупнейшими потребителями ИТ-услуг являются финансовый, производственный, государственный и телекоммуникационный секторы, суммарно обеспечившие более 65% ИТ-рынка.

В настоящее время наблюдается переход от индивидуального способа разработки программного обеспечения, к производственному. Программирование стало не ремеслом, а производством, что требует специфических специалистов по программной инженерии, квалифицированных кадров, способных эффективно участвовать в индустриальной реализации процессов разработки, эксплуатации и сопровождения программного обеспечения в качестве аналитиков, консультантов, интеграторов, спецификаторов, архитекторов, проектировщиков, менеджеров, разработчиков, тестеров, документаторов, инженеров по качеству и по безопасности ПО. Хотя надо отметить, что индивидуальный способ программирования и разработки не исчезает, просто его доля в общей массе программных разроботок сокращается.

Программная инженерия — сравнительно молодая научная дисциплина. Впервые термин Software Engineering был предложен в 1968 году на конференции НАТО, посвященной так называемому кризису программного обеспечения, возникшему с появлением вычислительной техники третьего поколения. Новая техника позволяла воплотить в жизнь не реализуемые ранее программные проекты. В результате программное обеспечение этих проектов достигло размеров и уровня сложности, намного превышающих аналогичные показатели у программных систем, реализованных на вычислительной технике предыдущих поколений. Возникла необходимость в новых технологиях и методах управления комплексными, сложными проектами разработки больших программных систем [2, 3].

В 1972 году IEEE выпустил первый номер Transactions on Software Engineering — труды по программной инженерии. Первый целостный взгляд на эту область профессиональной деятельности появился в 1979 году, когда компьютерное общество IEEE подготовило стандарт IEEE Std 730 по качеству программного обеспечения. В 1986 году IEEE выпустило IEEE Std 1002 Taxonomy of Software Engineering Standards. Наконец, в 1990 году началось планирование международных стандартов, в основу которых легли концепции и взгляды стандарта IEEE Std 1074 и результатов работы образованной в 1987 году совместной комиссии ISO/IEC JTC 1. В 1995 году группа этой комиссии SC7 Software Engineering выпустила первую версию международного стандарта ISO/IEC 12207 Software Lifecycle Processes (жизненный цикл программы). Этот стандарт стал первым опытом создания единого общего взгляда на программную инженерию. Издан соответствующий национальный стандарт России — ГОСТ Р ИСО/МЭК 12207–99 [4]. В свою очередь, IEEE и ACM, начав совместные работы еще в 1993 году с кодекса этики и профессиональной практики в данной области (ACM/IEEE-CS Software Engineering Code of Ethics and Professional Practice), к 2004 году сформулировали два ключевых описания того, что сегодня мы и называем основами программной инженерии — Software Engineering:

1) Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE 2004 Version — Руководство к Своду Знаний по Программной Инженерии.
2) Software Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering — Учебный план для преподавания программной инженерии в высших учебных заведениях.

***Программная инженерия*** *— это наука о систематизированных, регламентированных и квантифицируемых методах решения задач разработки, эксплуатации, сопровождения и утилизации программного обеспечения.*

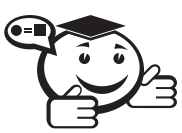
При этом как бизнес-процессы, так и программное обеспечение должны отвечать заданным техническим экономическим и социальным требованиям. Очевидно, что создание высококачественного ПО — очень трудоемкий процесс. Здесь должны быть задействованы необходимые для разработки процессы, инструментарии, технологии и человеческие ресурсы. В связи с этим возникла острая необходимость в специалистах, владеющих новыми технологиями и методами управления комплексными, сложными проектами разработки больших программных систем [5, 6].

Уникальность образовательного направления «программная инженерия» состоит в его тесном взаимодействии с наукой и бизнесом и практической значимости для экономики стран. Это определило его стремительное развитие во всем мире. Программная инженерия (или Software Engineering) как предмет преподавания является новым направлением, которое пока еще проходит процесс осмысления и опробирования в некоторых вузах России [7].

В последние годы программистское сообщество приложило немало усилий для упорядочения накопленных знаний и поиска способов их преобразования в учебные планы. В результате этих усилий появились проекты SWEBOK (Guide to the Software Engineering Body of Knowledge — «Руководство по сумме знаний о программной инженерии») и SE 2004 (Software Engineering 2004). SWEBOK отражает общепринятое представление о том, что должен знать программный инженер. В данном пособии излагаются кратко основные идеи, которые должен знать программист при разработке программного обеспечения, если придерживаться стратегии программной инженерии [8–11].

## Соглашения, принятые в книге

Для улучшения восприятия материала в данной книге используются пиктограммы и специальное выделение важной информации.



*Эта пиктограмма означает определение или новое понятие.*

Эта пиктограмма означает внимание. Здесь выделена важная информация, требующая акцента на ней. Автор здесь может поделиться с читателем опытом, чтобы помочь избежать некоторых ошибок.

## Контрольные вопросы по главе

# Глава 1

# ОСНОВНЫЕ ПОНЯТИЯ ПРОГРАММНОЙ ИНЖЕНЕРИИ

При обсуждении термина «программная инженерия» необходимо ответить на вопрос: «Что такое программное обеспечение (software)?».

***Программное обеспечение*** *— это программы, а также вся связанная с ними документация и конфигурационные данные, необходимые для корректной работы программы (т. е. не только программа).*

В конечном итоге, продаются не программы, а программные продукты, которые бывают двух типов: коробочные продукты (generic products or shrink-wrapped software) и заказные продукты (bespoke or customized products). Важная разница между ними заключается в том, кто ставит задачу (пишет спецификацию) [12].

***Программная инженерия*** *— это инженерная дисциплина, которая связана со всеми аспектами производства ПО от начальных стадий создания спецификации до поддержки системы после сдачи в эксплуатацию.*

В этом определении есть две важные части:

1) «Инженерная дисциплина». Инженеры добиваются результатов. Они применяют теории, методы и средства, пригодные для решения данной задачи, но они применяют их выборочно и всегда пытаются найти решения, даже в тех случаях, когда теорий или методов, соответствующих данной задаче, еще не существует.

2) Программные инженеры должны понимать значимость временных, финансовых и организационных ограничений. «Все аспекты производства ПО». Программная инженерия занимается не только техническими вопросами производства ПО, но и управлением программных проектов, а также разработкой средств, методов и теорий для поддержки процесса производства ПО.

Программные инженеры применяют систематичные и организованные подходы к работе для достижения максимальной эффективности и качества ПО. Общепринято мнение, что неправильно выбранный подход должен отрицательно сказаться на качестве и сроках сдачи системы. Но подход надо выбирать с умом — во многих случаях неформальные, «легкие» процессы могут оказаться значительно эффективнее (например, в таких задачах, как написание web-based applications или e-commerce systems).

С другой стороны, возникает вопрос: «В чем разница между программной инженерией (software engineering) и информатикой (computer science)?». Информатика занимается теорией и методами компьютерных и программных систем, в то время как программная инженерия занимается практическими проблемами создания ПО. Естественно, инженер-программист обязан в каком-то объеме знать информатику (точно так же, как электрический инженер должен в каком-то объеме знать физику). Однако объем теоретических знаний у разных специалистов сильно разнится и может быть очень невысоким, что тем не менее не мешает решать некоторые задачи (те, что попроще).

В идеале, вся программная инженерия должна быть поддержана какими-то теориями информатики, но на практике все значительно сложнее. Инженеры зачастую применяют первые попавшиеся методы, а элегантные теории информатики не всегда могут быть применены к реальным большим системам.

Программирование представляет сосбой непрерывный процесс создания программного обеспечния, и здесь возможен новый вопрос: «Что такое программный процесс?».

***Программный процесс*** *— это набор действий и связанных с ними результатов, приводящих к созданию программного продукта.*

Выделяют четыре основных фазы программного процесса:

1) Создание спецификации ПО (specification creation).
2) Разработка ПО (software development).
3) Тестирование ПО (включает в себя валидацию validation и верификацию verification).
4) Развитие или эволюция ПО (software evolution).

Для изучения любого процесса природы строят модель того или иного вида. Поэтому для процесса построения программы уместен вопрос: «Что такое модель программного процесса?».

***Модель программного процесса*** *— это упрощенное описание процесса разработки программы, представленное с некоторой точки зрения.*

Например:

- Модель технологического процесса (workflow model) — показывает последовательность действий, наряду со входами, выходами и зависимостями. Действия в этой модели представляют собой действия людей.
- Модель потоков данных (data flow or activity model) — представляет процесс в виде набора действий, каждый из которых выполняет некоторое преобразование данных. В этой модели действия могут быть более низкого уровня, чем в предыдущей модели (например, какие-то действия может выполнять компьютер).
- Модель роль/действие (role/action model) — показывает роли людей, участвующих в программном процессе, а также действия, за которые они отвечают.

Есть несколько традиционных моделей разработки ПО:

1) Водопадная модель — эволюционное развитие (в двух вариантах — с доработкой прототипа и с выбрасыванием, т. е. переписыванием).
2) Формальные преобразования — основаны на создании математических моделей системы и их дальнейшим преобразованиями, сохраняющими корректность.
3) Сборка из повторно используемых компонент — предполагает, что части системы уже существуют и процесс концентрируется скорее на интеграции, чем на разработке.

При разработке программного обеспечения необходимо оценивать стоимость, которую может приобрести продукт. Структура стоимости ПО может значительно отличаться для различных типов проектов и различных методологий. Для типичного проекта распределение стоимости выглядит приблизительно следующим образом:

1) Чем больше система и чем выше ее критичность, тем больший процент будет забирать тестирование.
2) Если же система разрабатывается с использованием эволюционного подхода, то трудно разделить на отдельные этапы создание спецификаций, проектирование и разработку. В результате, около 10% уходит на создание первичных высокоуровневых спецификаций, порядка 60% — на разработку и все остальное время уходит на тестирование.
3) Стоимость эволюции системы очень сильно варьируется в зависимости от размера системы и срока ее жизни. Во многих случаях стоимость сопровождения превосходит стоимость разработки в 3–4 раза.

4) При создании коробочных продуктов стоимость создания спецификации исчезающе мала (около 5%), на разработку (обычно эволюционным способом) уходит около 30–35% времени проекта, а все остальное — тестирование системы на всевозможных конфигурпциях. Для коробочных продуктов особенно трудно оценить стоимость эволюции — обычно процесс создания новой версии не имеет четко выраженного начала, и, кроме того, чаще всего из маркетинговых соображений новая версия подается не как модифицированный продукт, уже купленный пользователем, а как продукт абсолютно новый (хотя и совместимый со старыми).

Программные инженеры должны добиваться, чтобы анализ, спецификация, проектирование, разработка, тестирование и сопровождение программного обеспечения стали полезной и уважаемой профессией. В соответствии с их приверженностью к процветанию, безопасности и благополучию общества программные инженеры будут руководствоваться следующими Восемью Принципами:

1) *Общество*. Программные инженеры будут действовать соответственно общественным интересам.
2) *Клиент и работодатель*. Программные инженеры будут действовать в интересах клиентов и работодателя, соответственно общественным интересам.
3) *Продукт*. Программные инженеры будут добиваться, чтобы произведенные ими продукты и их модификации соответствовали высочайшим профессиональным стандартам.
4) *Суждение*. Программные инженеры будут добиваться честности и независимости в своих профессиональных суждениях.
5) *Менеджмент*. Менеджеры и лидеры программных инженеров будут руководствоваться этическим подходом к руководству разработкой и сопровождением ПО, а также будут продвигать и развивать этот подход.
6) *Профессия*. Программные инженеры будут улучшать целостность и репутацию своей профессии соответственно с интересами общества.
7) *Коллеги*. Программные инженеры будут честными по отношению к своим коллегам и будут всячески их поддерживать.
8) *Личность*. Программные инженеры в течение всей своей жизни будут учиться практике своей профессии и будут продвигать этический подход к практике своей профессии.

Полная версия кодекса: IEEE-CS/ACM Software Engineering Ethics and Professional Practices находится на сайте: http://www.computer.org/tab/seprof/code.htm#Public.

## Контрольные вопросы по главе 1

1) Что такое программное обеспечение (software)?
2) Определение программной инженерии.
3) Какие подходы применяют программные инженеры для достижения максимальной эффективности и качества ПО?
4) Каковы четыре основных фазы программного процесса?
5) Что такое модель программного процесса?
6) Что такое модель технологического процесса?
7) Что такое модель потоков данных?
8) Каковы традиционных моделей разработки ПО?
9) Чего программные инженеры должны добиваться при разработке ПО?
10) Укажите восемь Принципов, которыми программные инженеры будут руководствоваться при разработке ПО?

# Глава 2

# МЕТОДЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

***Метод программной инженерии*** *— это структурный подход к созданию ПО, нацеленный на создание эффективного продукта наиболее прибыльным (рентабельным, cost-effective) путем.*

Тема развивается с 1970-х: структурный анализ (1978), JSD (1983). Эти и другие методы тех времен были ориентированы на идентификацию основных функций системы; функционально-ориентированные методы до сих пор популярны и легли в основу всех современных идей автоматизированного программирования. Одна из современных технологий связана с возможностью автоматизированной разработки программ ПО (CASE), когда программист указывает лишь концепцию будущей программной системы и получает до 70% готового кода и далее остается лишь заполнить процедуры алгоритмами, которые ранее были обозначены лишь именами. CASE (Computer-Aided Software Engineering) включает в себя широкий комплекс программ, предназначенных для поддержки процессов создания программного продукта, включая анализ требований, моделирование, отладку и тестирование. Большинство современных методов поддержаны соответствующими CASE-средствами. В 1980-х годах эти методы были дополнены объектностью: Буч (1994), Рамбо (1991). Со временем эти методы были объединены в язык моделирования UML.

Практически все методы построены на идее создания графических моделей программной системы с последующим использованием этих моделей в качестве спецификации или архитектуры системы. Методы должны включать в себя следующие компоненты:

1) Описание моделей системы и их нотаций (например, объектные модели, конечно-автоматные модели и т. д.).
2) Правила, накладывающие ограничения на использование моделей в системе.

3) Рекомендации — эвристики, характеризующие хорошие приемы проектирования в данном методе (скажем, рекомендация о том, что ни у одного объекта не должно быть больше семи подобъектов).

4) Руководство к действию (наставление, process guidance) — описание действий, которым можно следовать во время создания моделей и последующего их использования (все атрибуты должны быть задокументированы до определения операций, связанных с этим объектом).

Нет идеальных методов, все они применимы только для тех или иных случаев. Например, объектно-ориентированные методы программирования хороши для интерактивных систем, но не для систем с жестким реальным временем. Здесь программист (программный инженер) должен обладать соответствующими знаниями для выбора того или иного метода, позволяющего эффективно решить поставленную задачу.

## Контрольные вопросы по главе 2

1) Что такое метод программной инженерии?
2) С какого времени начиналось становление программной инженерии?
3) Связана ли программная инженерия с методами автоматизированной разработки программ ПО (CASE)?
4) Может ли язык визуального моделирования ПО UML быть необходим программному инженеру?
5) Какие методы должны включать в себя четыре компоненты, необходимые для разработки ПО?

# Глава 3

# СВОЙСТВА ПРОГРАММЫ

Программные продукты обладают целым рядом свойств, не сводящихся к функциональности, которую они реализуют. Скорее, эти свойства относятся к поведению и внешнему виду программы во время выполнения, а также к структуре и организации исходной программы и связанной с ней документацией. Примерами подобных свойств (называемых также нефункциональными атрибутами) могут служить время отклика системы и читаемость исходных текстов. Конкретный набор свойств, которые следует ожидать от программы, конечно же, зависит от области ее применения: банковская система должна быть надежно защищенной, игра должна иметь малое время отклика, телефонная система коммутации должна быть надежной, научная программа — должна быть численно точной и т. п.

Свойства программы можно обобщить в несколько категорий:

1) Сопровождаемость (maintainability) — Система должна быть написана с расчетом на дальнейшее развитие. Это критическое свойство системы, т. к. изменения ПО неизбежны вследствие изменения бизнеса.
2) Надежность (dependability) — включает в себя отказоустойчивость, безопасность и защищенность. Надежное ПО не должно приводить к физическому или экономическому ущербу в случае сбоя системы.
3) Эффективность (efficiency) — ПО не должно впустую тратить системные ресурсы, такие, как память или процессорное время. Поэтому эффективность включает в себя отзывчивость, время процессора, утилизацию памяти и т. п.
4) Удобство использования (usability) — ПО должно быть легким в использовании, причем именно тем типом пользователей, на которых рассчитано приложение. Это включает в себя интерфейс пользователя и адекватную документацию.

Процесс выделения свойств программы зависит в основном от области действия программы и поэтому может иметь дополнительные характеристики. Однако указанные нами свойства будут верны для любой программы.

Известен факт, что программисты никогда не лишатся своего заработка по следующим причинам: разработка новой компьютерной техники (многоядерность, миниатюризация вычислительных устройств и др.), наличие ранее написанного программного кода (legacy challenge), который надо переписывать для новых систем, а также недостаток времени, отводимого на разработку программных продуктов (delivery challenge). Одновременно эти проблемы определяют основные трудности, которые стоят перед программной инженерией, — как развивиать преемственность между поколениями программ.

## Контрольные вопросы по главе 3

1) Какими свойствами обладают программные продукты?
2) Какие категории описывают свойства программы?
3) Что такое сопровождаемость ПО?
4) Что такое надежность ПО?
5) Что такое эффективность ПО?
6) Что такое Удобство использования ПО?
7) Какие причины приводят к тому, что программисты всегда будут с работой?

# Глава 4

# МЕТОДОЛОГИЯ MSF

Программная инженерия подразумевает разработку программы коллективом от постановки задачи (техническое задание) до сопровождения (работа с пользователем). Одним из способов преставления студентам методов программной инженерии является работа в рамках группового проектного обучения. Работа в коллективе специфична и требует от работающих определенных навыков (временные процессы, точность выполнения работ и др.). Существуют различные методологии (RUP, MSF, Prince2, XP и др.), позволяющие организовать работу проектных групп разной величины [19].

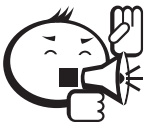
Рассмотрим методологию Microsoft Solutions Framework, поскольку:

- этот продукт является гибким и легко масштабируемым — его можно применять как для группы из трех программистов, так и в крупных коллективах из сотен и даже тысяч разработчиков,
- MSF является полностью бесплатным,
- существует перевод на русский язык официальной документации по MSF.

Немного об истории возникновения технологии программирования. В конце 60-х — начале 70-х годов прошлого века произошло событие, которое вошло в историю как первый кризис программирования. Событие состояло в том, что стоимость программного обеспечения стала приближаться к стоимости аппаратуры («железа»), а динамика роста этих стоимостей позволяла прогнозировать, что к середине 90-годов все человечество будет заниматься разработкой программ для компьютеров. Тогда и заговорили о программной инженерии (или технологии программирования) как о некоторой дисциплине, целью которой является сокращение стоимости программ.

С тех пор программная инженерия прошла достаточно бурное развитие. Этапы развития программной инженерии можно выделять по-разному. Каждый этап связан с появлением (или осознанием) очередной проблемы и нахождением путей и способов решения этой проблемы. Эти методы и по сей день составляют основу подходов к проектированию программных продуктов и базируются на пара-

дигмах программирования: процедурного, структурного, модульного и объектно-ориентированного.



Такие парадигмы, как процедурное и структурное программирование, легли в основу основных языков программирования.

Одной из проблем разработки программ в этом направлении была невозможность четко разделить функции коллектива разработчиков между задачами, и основной прчиной этого была проблема повторного использования кода. Эта проблема была частично решена в модульном и впоследствии в объектно-ориентированном программировании.

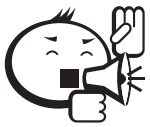
Со временем развития вычислительных устройств наблюдался рост сложности программ, что повлекло за собой развитие структурного программирования. На этом этапе наблюдается переход от разработки относительно простых программ к разработке сложных программных комплексов. К числу таких сложных программ относятся: системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т. д. Сложность таких комплексов оценивалась следующими показателями:

- Большой объем кода (миллионы строк).
- Большое количество связей между элементами кода.
- Большое количество разработчиков (сотни человек).
- Большое количество пользователей (сотни и тысячи).
- Длительное время использования.

Для таких сложных программ оказалось, что основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию. По аналогии с промышленной технологией стали говорить о жизненном цикле программного продукта как о последовательности определенных этапов: этапа проектирования, разработки, тестирования, внедрения и сопровождения. Задачи, которые возникли на этом этапе, связаны с развитием технологии, которая обеспечит «правильное» проектирование и кодирование. Основные принципы технологии структурного проектирования и кодирования:

- Нисходящее функциональное проектирование, при котором в системе выделяются основные функциональные подсистемы, которые потом разбиваются на подсистемы и т. д. (принцип «разделяй и властвуй»).
- Применение специальных языков проектирования и средств автоматизации использования этих языков.
- Дисциплина проектирования и разработки: планирование и документирование проекта, поддержка соответствия кода проектной документации.
- Структурное кодирование без goto.

Остановимся кратко на модульном программировании.



Главный принцип модульного программирования состоял в выделении набора процедур и оформлении их в виде модулей (например, процедуры графики, баз данных, работы с текстами и др.).

Каждый модуль снабжался описанием, в котором устанавливались правила его использования — интерфейс модуля. Интерфейс задавал связи модуля с основной программой — связи по данным и связи по управлению. При этом возможность повторного использования модулей определялась количеством и сложностью этих связей или тем, насколько эти связи удалось согласовывать с организацией данных и управления основной программы. Наиболее простыми в этом отношении оказались модули решения математических задач: решения уравнений, систем уравнений, задач оптимизации. К настоящему времени накоплены и успешно используются большие библиотеки таких модулей.

Для многих других типов модулей возможность их повторного использования оказалась проблематичной ввиду сложности их связей с основной программой. Повторное использование модулей со сложными интерфейсами является достаточно актуальным и по сей день. Для решения этой задачи разрабатываются специальные формы (структуры) представления модулей и организации их интерфейсов.

Следующим шагом в развитии подходов к программированию является объектно-ориентированное программирование (ООП). Следующая проблема разработки программ ярко обозначилась и остается до настоящего времени: изменения требований к программе стали возникать не только на стадии сопровождения, но и на стадии проектирования — проблема заказчика, который не знает, что он хочет. Создание программного продукта превратилось в его постоянное перепроектирование. Возник вопрос, как проектировать и писать программы, чтобы обеспечить возможность внесений изменений в программу, не меняя ранее написанного кода.

Решением этой проблемы стало использование подхода или метода, который стали называть объектно-ориентированным проектированием и программированием [22]. Суть подхода состоит в том, что вводится понятие класса как развитие понятия модуля с определенными свойствами и поведением, характеризующими обязанности класса. Каждый класс может порождать объекты — экземпляры данного класса. При этом работают основные принципы (парадигмы) ООП:

- Инкапсуляция — объединение в классе данных (свойств) и методов (процедур обработки).
- Наследование — возможность вывода нового класса из старого с частичным изменением свойств и методов.
- Полиморфизм — определение свойств и методов объекта по контексту.

Программная инженерия (или технология программирования) как некоторое направление возникло и формировалось под давлением роста числа проблем, возникающих при программировании, а также стоимости создаваемого программного обеспечения. Главная цель этой области знаний — сокращение стоимости и сроков разработки программ.

Программная инженерия прошла несколько этапов развития, в процессе которых были сформулированы фундаментальные принципы и методы разработки программных продуктов.

Основной принцип программной инженерии состоит в том, что программы создаются в результате выполнения нескольких взаимосвязанных этапов (анализ требований, проектирование, разработка, внедрение, сопровождение), составляющих жизненный цикл программного продукта.

Фундаментальными методами проектирования и разработки являются модульное, структурное и объектно-ориентированное проектирование и программирование.

## Контрольные вопросы по главе 4

1) Сможет ли один человек решить задачи программной инженерии?
2) Что такое методология Microsoft Solutions Framework?
3) В чем состоит сложность программных продуктов?
4) Укажите основные принципы технологии структурного проектирования и кодирования.
5) В чем главный принцип модульного программирования?
6) Основные признаки объектно-ориентированного подхода к программированию?
7) Суть объектно-ориентированного подхода?

# Глава 5

# СТАНДАРТИЗАЦИЯ И СТАНДАРТЫ

Как отмечалось, по происхождению программные продукты бывают двух типов: заказные (под заказ конкретного потребителя) и коробочные (для массовой продажи на рынке). Для заключения контракта заказчик должен быть уверен, что разработчик справится и не завалит проект. В мировой практике промышленного производства ответы на эти вопросы дают стандарты на производство продуктов и услуг и сертификация производителей на соответствие этим стандартам. Вопрос заказчика в этом случае звучит так: «Какими стандартами вы владеете и есть ли у вас сертификаты на соответствие этим стандартам?».

Процесс стандартизации и сертификации давно вошел и в программную инженерию, где он составляет основу промышленного производства программных продуктов. При изготовлении коробочных продуктов стандартизация имеет не меньшее значение, т. к. она обеспечивает качество продуктов и продвижение их на рынок. При этом применяется некоторая технология производства. Эта технология должна соответствовать стандартам на товары или услуги. Применяемая организацией технология проходит сертификацию на соответствие этим стандартам [23, 24].

*Технология* (происходит от греческого *téchne* — искусство, мастерство) определяется как совокупность приёмов и способов получения, обработки или переработки сырья, материалов, полуфабрикатов или изделий, осуществляемых в различных отраслях промышленности, в строительстве и т. д.

***Стандарт*** *(происходит от английского standard — норма, образец, мерило) — это утверждаемый компетентным органом нормативно-технический документ, устанавливающий комплекс норм, правил по отношению к предмету стандартизации, типовой образец, эталон, модель, принимаемые за исходные для сопоставления с ними других предметов.*

Например: ГОСТ ЕСПД — единая система программной документации — документы, описывающие состав и структуру документации на разработку программ для ЭВМ (общее описание, техническое задание, эскизный проект, технический проект, описание применения). Типовые образцы — эталоны мер и весов (эталон метра, хранящийся в Париже в палате мер и весов).

Сертификация в переводе с латыни означает «сделано верно». Для того чтобы убедиться в том, что продукт «сделан верно», надо знать: каким требованиям он должен соответствовать, каким образом возможно получить достоверные доказательства этого соответствия. Общепризнанным способом такого доказательства служит сертификация соответствия и заявление о соответствии.

Среди всего многообразия стандартов принято выделять следующие основные типы стандартов:

*Корпоративные стандарты* разрабатываются крупными фирмами (корпорациями) с целью повышения качества своей продукции. Такие стандарты разрабатываются на основе собственного опыта и с учетом требований мировых стандартов. Корпоративные стандарты не сертифицируются, но являются обязательными для применения внутри корпорации. В условиях рыночной конкуренции могут иметь закрытый характер, напрмиер в IT-сфере известны корпоративные стандарты, разработанные фирмами Microsoft, Intel, IBM.

*Отраслевые стандарты* действуют в пределах организаций некоторой отрасли (министерства). Например, СНИП — строительные нормы и правила. Разрабатываются с учетом требований мирового опыта и специфики отрасли. Являются, как правило, обязательными для отрасли. Подлежат сертификации.

*Государственные стандарты* (ГОСТы) принимаются государственными органами, имеют силу закона. Разрабатываются с учетом мирового опыта или на основе отраслевых стандартов. Могут иметь как рекомендательный, так и обязательный характер (стандарты безопасности). Для сертификации создаются государственные или лицензированные органы сертификации.

*Международные стандарты*. Разрабатываются, как правило, специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов. Имеют сугубо рекомендательный характер. Право сертификации получают организации (государственные и частные), прошедшие лицензирование в международных организациях.

Кто разрабатывает стандарты программной инжерении SE? Основными разработчиками международных стандартов являются следующие организации:

*ISO — International Organization for Standardization* — Международная организация по стандартизации. Наиболее представительная и влиятельная организация, разрабатывающая стандарты почти во всех областях деятельности, в том числе и в IT. Разработка международных стандартов обычно осуществляется техническими комитетами ISO. Каждый *комитет-член*, заинтересованный в деятельности, для которой создан технический комитет, имеет право быть представленным в этом комитете.

Международные правительственные и неправительственные организации, имеющие связи с ISO, также принимают участие в работах. Стандарты ISO являются рекомендательными. Как правило, никакого контроля за выполнением стандартов, никакой сертификации на соответствие своим стандартам ISO не ведет — это так-

же считается суверенным правом стран. Обычно эти процедуры поручаются либо специально назначенному государственному органу регистрации, либо так называемой третьей стороне — лаборатории или аудиторскому институту, в том числе и частному аудитору, действующему на коммерческой основе.

*ACM — Association for Computing Machinery* (http://www.acm.org/) — Ассоциация по вычислительной техники. Всемирная научная и образовательная организация в области вычислительной техники. Известна также и разработкой образовательных стандартов. ACM — Association for Computing Machinery — название почти никогда не переводится. Можно перевести как Ассоциация для вычислительной техники, что звучит весьма коряво. ACM является крупнейшей всемирной научной и образовательной организацией, объединяющей более 75000 профессионалов компьютерной науки. Основанная в 1947 г, ACM ежегодно проводит до 100 международных (научных и практических) конференций, издает несколько десятков научных журналов и присуждает большое количество авторитетных наград за достижения в области компьютерной науки, в т.ч. A.M. Turing Award, известную как «нобелевская премия информатики». Под эгидой ACM проводятся ежегодные международные студенческие олимпиады по программированию.

*SEI — Software Engineering Institute* — Институт Программной Инженерии (www.sei.cmu.edu). Исследования в области программной инженерии с упором на разработку методов оценки и повышения качества ПО. Стандарты по качеству ПО и зрелости организаций, разрабатывающих ПО. SEI ставит своей основной задачей создание методик для оценки уровня развития внутренних процессов в организации. В качестве подразделения, широко известного благодаря разработкам в области вычислительной техники и программного инжиниринга, SEI имеет доступ к самым передовым техническим инновациям. С 1984 года SEI развивает и пропагандирует методики для разработки высококачественного ПО. Первая версия Модели Технологической Зрелости Компании-Разработчика ПО (Capability Maturity Model for Software, SW-CMM) была создана в SEI в 1991 году.

*PMI — Project Management Institute* — Международный Институт Проектного Менеджмента (http://www.pmi.ru). Некоммерческая организация, целью которой является продвижение, пропаганда, развитие проектного менеджмента в разных странах. PMI разрабатывает стандарты проектного менеджмента, занимается повышением квалификации специалистов.

PMI являлся ведущей профессиональной организацией по управлению проектами в таких областях, как авиакосмическая и автомобильная промышленность, управление коммерческими предприятиями, машиностроение, финансовые операции, информационные технологии, фармацевтика, телекоммуникации и многие другие. PMI предоставляет всеобъемлющее руководство по разработке стандартов для проектного менеджмента (стандарт по управлению проектами PMBOK). PMI стал первой организацией в мире, имеющей программу сертификации специалистов по управлению проектами — Project Management Professional (PMP).

Для обучения проектному менеджменту и подготовки к экзамену PMP созданы Registered Education Provider (R.E.P) — Сертифицированный провайдер по образованию — во многих странах мира. Исследования в области проектного менеджмента поддерживаются за счет проведения конференций, предоставления грантов, выпуска научных трудов, создания исследовательской базы данных и т. д. Кроме то-

го, собирается и сортируется информация о текущем состоянии дел, потребностях, накопленных знаниях по проектному менеджменту, и на этой основе оценивается будущее профессии и путь ее развития.

PMI выпускает три вида периодических изданий для индивидуальных лиц, занимающихся проектным менеджментом: ежемесячный журнал PM Network, ежеквартальный журнал Project Management Journal и ежемесячный информационный бюллетень PMI Today. PMI является ведущим мировым издателем литературы и учебных материалов по проектному менеджменту. В онлайновом магазине PMI в настоящее время доступно более 1000 наименований.

*IEEE* — Институт инженеров по электронике (http://www.ieee.org и http://www.computer.org.ru/). Поддержка научных и практических разработок в области электроники и вычислительной техники. Большие вложения в разработку стандартов в этой области. IEEE объединяет почти 400000 технических специалистов из более чем 150 стран. IEEE состоит из ряда профессиональных сообществ, в самое крупное из которых — IEEE Computer Society — входят более 100000 человек. Компьютерное сообщество IEEE ежегодно спонсирует около ста пятидесяти научных конференций и симпозиумов, публикует более 20 периодических изданий. IEEE Computer Society также широко известно своей деятельностью по стандартизации, которую на сегодняшний день в рамках сообщества осуществляют порядка 200 рабочих групп.

Основные стандарты программной инженерии, наиболее известные в мире:

*ISO/IEC 12207 — Information Technology — Software Life Cycle Processes* — Процессы жизненного цикла программных средств. Стандарт содержит определения основных понятий программной инженерии (в частности программного продукта и жизненного цикла программного продукта), структуры жизненного цикла как совокупности процессов, детальное описание процессов жизненного цикла.

*SEI CMM — Capability Maturity Model (for Software)* — модель зрелости процессов разработки программного обеспечения. Стандарт отвечает на вопрос: «Какими признаками должна обладать профессиональная организация по разработке ПО?». Профессионализм организации определяется через зрелость процесса, применяемого этой организацией. Выделяются пять уровней зрелости процесса.

*ISO/IEC 15504 — Software Process Assessment* — Оценка и аттестация зрелости процессов создания и сопровождения ПО. Является развитием и уточнением ISO 12207 и SEI CMM. Содержит расширенное по отношению ISO 12207 количество процессов жизненного цикла и 6 уровней зрелости процессов. Дается подробное описание схемы аттестации процессов, на основе результатов которой может быть выполнена оценка зрелости процессов и даны рекомендации по их усовершенствованию.

*PMBOK — Project Management Body of Knowledge* — Свод знаний по управлению проектами. Содержит описания состава знаний по следующим 9 разделам (областям знаний) управления проектами.

*SWBOK — Software Engineering Body of Knowledge* — Свод знаний по программной инженерии — содержит описания состава знаний по 10 разделам (областям знаний) программной инженерии.

*ACM/IEEE CC2001 — Computing Curricula 2001* — Академический образовательный стандарт в области компьютерных наук. Выделены 4 основных раздела ком-

пьютерных наук: Computer science, Computer engineering, Software engineering и Information systems, по каждому из которых описаны области знаний соответствующего раздела, состав и планы рекомендуемых курсов.

*Понятие **жизненного цикла ПО** как некоторой последовательности этапов, которые надо выполнить для создания ПО (проектирование, разработка, тестирования), составляет одно из фундаментальных понятий программной инженерии.*

Понятие возникло в конце 60-х годов, когда разработкой ПО занималось много фирм и различными разработчиками трактовалось по-разному. Путаница в терминологии порождала много проблем во взаимопонимании между разработчиками, между разработчиками и заказчиками. Необходим был стандарт на представление терминов, понятий и составных элементов жизненного цикла ПО. Этот стандарт был принят ISO в 1995 году. Следует отметить, что работы по нему были начаты в 1987 году и стандарт формировался взаимными усилиями 125 стран — участниц ISO. В России этот стандарт принят в 2000 году под названием «ГОСТ Р ИСО/МЭК 12207 Процессы жизненного цикла программных средств».

*В стандарте ISO 12207 дается определение **программного продукта (ПО)** как набора компьютерных программ, процедур и связанной с ними документации и данных и определение жизненного цикла программного продукта как непрерывного процесса, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации (см. ГОСТ ИСО 12207: http://www.staratel.com/iso/InfTech/DesignPO/ISO12207/ISO12207–99/ISO12207.htm).*

Для тех же процессов существует стандарт *SEI CMM — Capability Maturity Model (for Software)* — модель зрелости процессов разработки программного обеспечения — известный отчет SEI, который формально стандартом не является, но приобрел характер международного стандарта в силу его интересности и практической полезности. Отчет появился в 1993 году как результат исследования на тему: «Как выбирать организацию, которой можно доверить выполнение крупного IT проекта?». Это исследование проводилось SEI по заказу министерства обороны США, которое было очень озадачено этой проблемой. В отчете изложена модель зрелости организаций, которая определялась через зрелость процесса разработки ПО, применяемого в этой организации. В этой модели выделяется пять уровней зрелости процесса, которые и устанавливают степень готовности организации выполнить крупный проект:

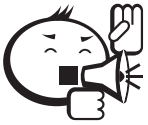
1) Начальный (Initial) — Технология полностью импровизированная, в некоторых случаях — даже хаотическая. Успех всецело зависит от усилий отдельных сотрудников.
2) Повторяемый (Repeatable) — Базовые процессы управления проектом ПО установлены. Есть дисциплина соблюдения, что обеспечивает возможность повторения успеха предыдущих проектов в той же прикладной области.
3) Определенный (Defined) — Процессы задокументированы, стандартизованы и интегрированы в единую для всей организации технологию создания ПО. Для каждого проекта используется адаптивный вариант этой технологии.
4) Управляемый (Managed) — Собираются и накапливаются метрики (объективные данные) о качестве исполнения процессов и выходной продукции. Управление процессами и выходной продукцией осуществляется по количественным оценкам.
5) Оптимизируемый (Optimized) — Совершенствование технологии создания ПО осуществляется непрерывно на основе количественной обратной связи от процессов и пилотного внедрения инновационных идей.

*PMI PMBOK — Project Management Body of Knowledge*, Свод знаний по управлению проектами. Последняя версия стандарта вышла в 2004 году. Содержит описания состава знаний по следующим 9 разделам (областям знаний) управления проектами:

1) Управление интеграцией — Project Integration Management.
2) Управление ограничениями — Project Scope Management.
3) Управление временем — Project Time Management.
4) Управление затратами — Project Cost Management.
5) Управление рисками — Project Risk Management.
6) Управление персоналом — Project Personnel Management.
7) Управление коммуникациями — Project Communication Management.
8) Управление закупками — Project Procurement Management.
9) Управление качеством — Project Quality Management.

Подробнее информацию можно найти на сайте http://www.tline.ru/library/.

Наиболее известный подход к разработке программного обеспечения разработан в IEEE. Основополагающий документ именуется *IEEE Computer Society Software Engineering Body of Knowledge* — Свод знаний по программной инженерии — проект IEEE Computer Society. Официальная версия вышла в 2004 г.



Основная идея проекта заключается в создании некоторого базового набора общепринятых знаний, необходимых любому профессиональному программисту.

Содержит описания состава знаний по следующим 10 разделам (областям знаний) программной инженерии:

- Software Requirements — требования к ПО.
- Software Design — проектирование ПО.
- Software Construction — конструирование ПО.
- Software Testing — тестирование ПО.
- Software Maintenance — сопровождение ПО.
- Software Configuration Management — управление конфигурациями.
- Software Engineering Management — управление IT проектом.
- Software Engineering Process — процесс программной инженерии.
- Software Engineerting Tools and Methods — методы и инструменты.
- Software Quality — качество ПО.

Подробнее информацию можно найти на сайте: http://www.swebok.org/.

Кроме технологических задач, решаются и задачи подготовки кадров по внедрению знаний программной инженерии и продвижению этих знаний на рынок. Известный подход двух разработчиков ACM и IEEE — Computing Curricula позволяет сформировать образовательный стандарт к подготовке прораммных инженеров. ACM/IEEE Computing Curricula 2001 — Академический образовательный стандарт в области компьютерных наук — совместный проект международных профессиональных обществ ACM и IEEE Computer Society. Основная идея проекта состоит в разработке стандартов на учебные курсы по компьютерным наукам. В стандарте 2001 года выделены 4 основных раздела компьютерных наук:

- Computer science — Информатика (2001г); http://se.math.spbu.ru/cc2001;
- Computer engineering — Компьютерная инженерия;
- Software engineering — Программная инженерия (2004г.);
- Information systems — Информационные системы.

Работа над проектом продолжается, и рабочие материалы можно посмотреть на сайте: http://www.computer.org/education/cc2001. По содержанию образовательные стандарты состоят из описания областей знаний соответствующего раздела, состава и планов рекомендуемых курсов. Областями знаний раздела Software engineering являются:

- Computing Essentials — Основы применения ЭВМ.
- Mathematical & Engineering Fundamentals — Математические и инженерные основы.
- Professional Practice — Профессиональная практика.
- Software Modeling & Analysis — Моделирование и анализ ПО.
- Software Design — Проектирование ПО.
- Software V & V — Верификация и валидация ПО.
- Software Evolution — Эволюция ПО.

- Software Process — Процесс ПО.
- Software Quality — Качество ПО.
- Software Management — Управление проектом.

Текст ACM/IEEE Computing Curricula 2001 переведен на русский язык, его можно найти на сайте http://se.math.spbu.ru/cc2001.

## Контрольные вопросы по главе 5

1) Что такое технология, стандарт, сертификация?
2) Что такое корпоративные, отраслевые, государственные стандарты?
3) Что содержит ISO/IEC 12207 — Процессы жизненного цикла программных средств?
4) На какой вопрос отвечает стандарт SEI CMM — модель зрелости процессов разработки программного обеспечения?
5) На что нацелен стандарт ISO/IEC 15504 Оценка и аттестация зрелости процессов создания и сопровождения ПО?
6) Что такое PMBOK — Свод знаний по управлению проектами?
7) Что такое SWEBOK — Свод знаний по программной инженерии?
8) Зачем нужен стандарт ACM/IEEE CC2001 — Академический образовательный стандарт в области компьютерных наук?
9) Укажите пять уровней зрелости процесса разработки ПО.
10) Укажите 10 разделов состава знаний программной инженерии.

# Глава 6

# ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ПРОДУКТА

Немного истории. Появление понятия жизненного цикла ПО было связано с кризисом программирования, который наметился в конце 60-х — начале 70-х годов прошлого века. Суть кризиса состояла в том, что программные проекты все чаще стали выходить из-под контроля: нарушались сроки, превышались запланированные объемы финансирования, результаты не соответствовали требуемым. Многие проекты вообще не доводились до завершения. Кроме того, оказалось, что недостаточно разработать программу, а надо ее еще сопровождать и этап сопровождения часто требует больше средств, чем разработка [15–18].

Ситуация была вызвана ростом сложности проектов. Необходимо было принимать меры для радикального усовершенствования принципов и методов разработки ПО с учетом его развития и сопровождения. Заговорили о том, что надо обратиться к опыту промышленного проектирования и производства, где был накоплен опыт успешной разработки не менее сложных проектов.

Впервые о жизненном цикле ПО заговорили в 1968 г. в Лондоне, где состоялась встреча большого количества руководителей проектов по разработке ПО. На встрече анализировались проблемы и перспективы проектирования, разработки, распространения и поддержки программ. Применяющиеся принципы и методы разработки ПО требовали постоянного усовершенствования. Именно на этой встрече была предложена концепция жизненного цикла ПО (SLC — Software Lifetime Cycle) как последовательности шагов-стадий, которые необходимо выполнить в процессе создания и эксплуатации ПО.

Методологическую основу промышленной инженерии составляет понятие жизненного цикла изделия (продукта) как совокупности всех действий, которые надо выполнить на протяжении всей «жизни» программного изделия. Смысл жизненного цикла состоит во взаимосвязанности всех этих действий. Итак, жизненный цикл промышленного изделия: последовательность этапов [(фаз, стадий): проектирования, изготовления образца, организации производства, серийного производства,

эксплуатации, ремонта, вывода из эксплуатации], состоящих из технологических процессов, действий и операций.

Организация промышленного производства с позиции жизненного цикла позволяет рассматривать все его этапы во взаимосвязи, что ведет к сокращению сроков, стоимости и трудозатрат.

Типовая модель процессов жизненного цикла програмной системы начинается с концепции идеи системы или потребности в ней, охватывает проектирование, разработку, применение и сопровождение системы и заканчивается снятием системы с эксплуатации. Программные средства служат для выполнения определенных функций систем на компьютерах. Модель жизненного цикла системы обычно разделяют на последовательные периоды реализации — стадии или этапы.

Каждый подобный период включает основные реализуемые в нем процессы, работы и задачи, при завершении которых может потребоваться переход к следующему периоду реализации. Общую модель жизненного цикла сложной системы обычно разделяют на следующие основные этапы с последующей адаптацией каждого из них в модели жизненного цикла конкретной системы:

- определение потребностей;
- исследование и описание основных концепций;
- проектирование и разработка;
- испытания системы;
- создание и производство;
- распространение и продажа;
- эксплуатация;
- сопровождение и мониторинг;
- снятие с эксплуатации (утилизация).

По особенностям и свойствам жизненного цикла программ их целесообразно делить на ряд классов и категорий, из которых наиболее различающимися являются два крупных класса — малые и большие.

Первый класс составляют создаваемые одиночками или небольшими коллективами (3–5) специалистов относительно небольшие программы, которые:

- создаются преимущественно для получения конкретных результатов автоматизации научных исследований или для анализа относительно простых процессов самими разработчиками программ;
- не предназначены для массового тиражирования и распространения как программного продукта на рынке, их оценивают качественно и интуитивно преимущественно как «художественные произведения»;
- не имеют конкретного независимого заказчика-потребителя, определяющего требования к программам и их финансирование;
- не ограничиваются заказчиком допустимой стоимостью, трудоемкостью и сроками их создания, требованиями заданного качества и документирования;

- не подлежат независимому тестированию, гарантированию качества и/или сертификации.

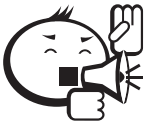
Для таких, а также для многих других видов относительно не сложных программ нет необходимости в регламентировании их жизненного цикла, в длительном применении и сопровождении множества версий, в формализации и применении профилей стандартов и сертификации качества программ. Их разработчики не знают и не применяют регламентирующих, нормативных документов, вследствие чего жизненный цикл таких изделий имеет непредсказуемый характер по структуре, содержанию, качеству и стоимости основных процессов «творчества».

Второй класс составляют крупномасштабные комплексы программ для сложных систем управления и обработки информации, оформляемые в виде программных продуктов с гарантированным качеством и отличающиеся следующими особенностями и свойствами их жизненного цикла:

- большая размерность, высокая трудоемкость и стоимость создания таких комплексов программ определяют необходимость тщательного анализа экономической эффективности всего их жизненного цикла и возможной конкурентоспособности на рынке;
- от заказчика, финансирующего проект программного средства и/или базы данных, разработчикам необходимо получать квалифицированные конкретные требования к функциям и характеристикам проекта и продукта, соответствующие выделенному финансированию и квалификации исполнителей проекта;
- для организации и координации деятельности специалистов-разработчиков при наличии единой, крупной целевой задачи создания и совершенствования программного продукта необходимы квалифицированные менеджеры проектов;
- в проектах таких сложных программных средств и баз данных с множеством различных функциональных компонентов участвуют специалисты разной квалификации и специализации, от которых требуется высокая ответственность за качество результатов деятельности каждого из них;
- от разработчиков проектов требуются гарантии высокого качества, надежности функционирования и безопасности применения компонентов и поставляемых программных продуктов, в которые не допустимо прямое вмешательство заказчика и пользователей для изменений, не предусмотренных эксплуатационной документацией разработчиков;
- необходимо применять индустриальные, регламентированные стандартами процессы, этапы и документы, а также методы, методики и комплексы средства автоматизации, технологии обеспечения жизненного цикла комплексов программ.

Такие крупномасштабные комплексы программ являются компонентами систем, реализующими обычно их основные, функциональные свойства, увеличивающими сложность и создающими предпосылки для последующих изменений их жизненного цикла. Реализация ЖЦ, методологии управления и изменения ПС зависит от многих факторов: от персонала, технических, организационных и дого-

ворных требований и сложности проекта. Множество текущих состояний и модификаций компонентов сложных ПС менеджерам необходимо упорядочивать, контролировать их развитие и применение участниками проекта.



Организованное, контролируемое и методичное отслеживание динамики изменений в жизненном цикле программ и данных, их слаженная разработка при строгом учете и контроле каждого изменения являются основой эффективного, поступательного развития каждой крупной системы методами программной инженерии.

Разрабатывались стандарты ЖЦ ПО. Наиболее известными являются:

*1995 г. IEEE 1074. Процессы жизненного цикла для развития программного обеспечения*. Охватывает полный жизненный цикл ПС, в котором выделяются шесть крупных базовых процессов. Эти процессы детализируются 16 частными процессами. В последних имеется еще более мелкая детализация в совокупности на 65 процессов-работ. Содержание каждого частного процесса начинается с описания общих его функций и задач и перечня действий — работ при последующей детализации. Для каждого процесса в стандарте представлена входная и результирующая информация о его выполнении и краткое описание сущности процесса. В стандарте внимание сосредоточено преимущественно на непосредственном создании ПС и на процессах предварительного проектирования. В приложении представлены четыре варианта адаптации максимального состава компонентов ЖЦ ПС к конкретным особенностям типовых проектов.

*ISO 12207 (15504) Жизненный цикл ПП: структура и организация.* Разрешением проблем стандартизации ЖЦ ПО явились разработка и принятие в 1995 г. стандарта ISO/IEC 12207 — Information Technology — Software Life Cycle Processes (ISO — International Organization of Standardization — Международная организация по стандартизации; IEC — International Electrotechnical Commission — Международная электротехническая комиссия). В 2000 г. он был принят как ГОСТ 12207. Процессы жизненного цикла программных средств.

Стандарт ISO 12207 разрабатывался с учетом лучшего мирового опыта на основе вышеперечисленных стандартов. Основными результатами стандарта ISO 12207 являются:

- Введение единой терминологии по разработке и применению ПО (предназначен не только для разработчиков, но и для заказчиков, пользователей, всех заинтересованных лиц).
- Разделение понятий ЖЦ ПО и модели ЖЦ ПО. ЖЦ ПО в стандарте вводится как полная совокупность всех процессов и действий по созданию и применению ПО, а модель ЖЦ — конкретный вариант организации ЖЦ, обоснованно (разумно) выбранный для каждого конкретного случая.
- Описание организации ЖЦ и его структуры (процессов).
- Выделение процесса адаптации стандарта для построения конкретных моделей ЖЦ.

В стандарте ISO 12207 даются следующие определения:

- *Программный продукт (software product):* Набор машинных программ, процедур и, возможно, связанных с ними документации и данных.
- *Жизненный цикл программного продукта (software life cycle)* — это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.
- *Процесс (process):* Набор взаимосвязанных работ, которые преобразуют исходные данные в выходные результаты.

Стандарт определяет организацию ЖЦ программного продукта как совокупность процессов, каждый из которых разбит на действия, состоящие из отдельных задач. Устанавливает структуру (архитектуру) ЖЦ программного продукта в виде перечня процессов, действий и задач, которые делятся на три группы:

1) Основные.
2) Вспомогательные.
3) Организационные.

Отдельно описан процесс адаптации стандарта, содержащий основные работы, которые должны быть выполнены при адаптации настоящего стандарта к условиям конкретного программного проекта. К числу основных относятся процессы:

1) *Заказа.* Определяет работы заказчика, то есть организации, которая приобретает систему, программный продукт или программную услугу.
2) *Поставки.* Определяет работы поставщика, то есть организации, которая поставляет систему, программный продукт или программную услугу заказчику.
3) *Разработки.* Определяет работы разработчика, то есть организации, которая проектирует и разрабатывает программный продукт.
4) *Эксплуатации.* Определяет работы оператора, то есть организации, которая обеспечивает эксплуатационное обслуживание вычислительной системы в заданных условиях в интересах пользователей.
5) *Сопровождения.* Определяет работы персонала сопровождения, то есть организации, которая предоставляет услуги по сопровождению программного продукта, состоящие в контролируемом изменении программного продукта с целью сохранения его исходного состояния и функциональных возможностей. Данный процесс охватывает перенос и снятие с эксплуатации программного продукта.

*Вспомогательными являются процессы:*

- Документирования. Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.
- Управления конфигурацией. Определяет работы по управлению конфигурацией.
- Обеспечения качества. Определяет работы по объективному обеспечению того, чтобы программные продукты и процессы соответствовали требованиям, установленным для них, и реализовывались в рамках утвержденных

планов. Совместные анализы, аудиторские проверки, верификация и аттестация могут использоваться в качестве методов обеспечения качества.

- Верификации. Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.
- Аттестации. Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.
- Совместного анализа. Определяет работы по оценке состояния и результатов какой-либо работы. Данный процесс может использоваться двумя любыми сторонами, когда одна из сторон (проверяющая) проверяет другую сторону (проверяемую) на совместном совещании.
- Аудита. Определяет работы по определению соответствия требованиям, планам и договору. Данный процесс может использоваться двумя сторонами, когда одна из сторон (проверяющая) контролирует программные продукты или работы другой стороны (проверяемой).
- Решения проблем. Определяет процесс анализа и устранения проблем (включая несоответствия), независимо от их характера и источника, которые были обнаружены во время осуществления разработки, эксплуатации, сопровождения или других процессов.

*Организационные процессы жизненного цикла:*

- Управления. Определяет основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла.
- Создания инфраструктуры. Определяет основные работы по созданию основной структуры процесса жизненного цикла.
- Усовершенствования. Определяет основные работы, которые организация (заказчика, поставщика, разработчика, оператора, персонала сопровождения или администратора другого процесса) выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла.
- Обучения. Определяет работы по соответствующему обучению персонала.

Стандарт ISO 12207 определяет общие правила разработки программного обеспечения. С развитием вычислительной техники и технологий программирования появилась необходимость разработки усовершенствованного стандарта. В 1998 г. выходит новый стандарт ISO/IEC TR 15504: Information Technology — Software Process Assessment (Оценка процессов разработки ПО). В этом документе рассматриваются вопросы аттестации, определения зрелости и усовершенствования процессов жизненного цикла ПО. Связь со стандартом ISO 12207 состоит в том, что все процессы стандарта ISO 15504 принадлежат к одному из следующих типов:

- базовый — процесс из 12207;
- расширенный — расширение процесса из 12207;
- новый — процесс, не описанный в 12207;
- составляющий — часть процесса из 12207;
- расширенный составляющий — расширенная часть процесса из 12207.

В соответствии с новой классификацией в трех группах процессов вводятся пять категорий процессов:

Основные процессы:

1. CUS: Потребитель-поставщик.

2. ENG: Инженерная.

Вспомогательные процессы:

3. SUP: Вспомогательная.

Организационные процессы:

4. MAN Управленческая.

5. ORG: Организационная.

Инженерная категория процессов состоит из процессов, которые непосредственно определяют, реализуют или поддерживают программный продукт, его взаимодействие с системой и документацию на него. В тех случаях, когда система целиком состоит из программных средств, инженерные процессы имеют отношение только к созданию и поддержанию этих программных средств. Включает следующие процессы:

- ENG.1 Процесс разработки (Development process).
- ENG.1.1 Процесс анализа требований и разработки системы (Systemrequirements analysis and design process).
- ENG.1.2 Процесс анализа требований к программным средствам (Software requirements analysis process).
- ENG.1.3 Процесс проектирования программных средств (Software design process).
- ENG.1.4 Процесс конструирования программных средств (Software construction process).
- ENG.1.5 Процесс интеграции программных средств (Software integration process).
- ENG.1.6 Процесс тестирования программных средств (Software testing process).
- ENG.1.7 Процесс интеграции и тестирования системы (System integration andtesting process).
- ENG.2 Процесс сопровождения системы и программных средств (System and software maintenance process).

Вспомогательная категория состоит из процессов, которыми могут пользоваться любые другие процессы (включая другие вспомогательные процессы) в различные моменты жизненного цикла программных средств.

Включает следующие процессы:

- SUP.1 Процесс документирования (Documentation process).
- SUP.2 Процесс управления конфигурацией (Configuration management process).
- SUP.3 Процесс обеспечения качества (Quality assurance process).
- SUP.4 Процесс верификации (Verification process).
- SUP.5 Процесс проверки соответствия (Validation process).

- SUP.6 Процесс совместных проверок (Joint review process).
- SUP.7 Процесс аудита (Audit process).
- SUP.8 Процесс разрешения проблем (Problem resolution process).

Управленческая категория состоит из процессов, содержащих практики общего характера, которые могут быть использованы каждым, кто управляет любым проектом или процессом в ходе жизненного цикла программных средств.

К управленческой категории относятся следующие процессы:

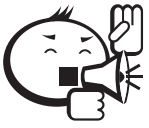
- MAN.1 Процесс административного управления (Management process).
- MAN.2 Процесс управления проектами (Project management process).
- MAN.3 Процесс управления качеством (Quality Management process).
- MAN.4 Процесс управления рисками (Risk Management process).

Организационная категория процессов состоит из процессов, которые устанавливают цели функционирования организации и создают активы процессов, продуктов и ресурсов, которые, будучи использованы в проектах организации, способствуют выполнению ее целей. Хотя организационные практики в целом относятся не только к процессам, относящимся к программным средствам, последние выполняются в общем контексте организации, и для их эффективного использования необходимо соответствующее окружение. Кратко, эти организационные процессы:

- создают инфраструктуру организации;
- используют все лучшее из того, что имеется (передовой опыт) во всех частях организации (эффективные процессы, лучшие навыки, качественный программный код, хорошие средства поддержки);
- делают это общедоступным в рамках всей организации;
- создают базу для постоянного совершенствования во всей организации.

Организационной категории принадлежат процессы:

- ORG.1 Процесс организационных установок (Organizational alignment process).
- ORG.2 Процесс усовершенствования (Improvement process).
- ORG.2.1 Процесс создания процессов (Process establishment process).
- ORG.2.2 Процесс аттестации процессов (Process assessment process).
- ORG.2.3 Процесс усовершенствования процессов (Process improvement process).
- ORG.3 Процесс административного управления кадрами (Human resource management process).
- ORG.4 Процесс создания инфраструктуры (Infrastructure process).
- ORG.5 Процесс измерения (Measurement process).
- ORG.6 Процесс повторного использования (Reuse process).



Модель жизненного цикла ПО описывает набор фаз (этапов, стадий) проекта по созданию ПО, в которых выполняются отдельные процессы, разбитые на операции и задачи.

В глоссарии PMI (PMI. Глоссарий http://www.pmi.ru/glossary/) даются следующие определения этих понятий:

*Жизненный цикл проекта*. Набор обычно последовательных фаз проекта, количество и состав которых определяется потребностями управления проектом организацией или организациями, участвующими в проекте.

*Фаза проекта.* Объединение логически связанных операций проекта, обычно завершающихся достижением одного из основных результатов.

*Процесс.* Набор взаимосвязанных ресурсов и работ, благодаря которым входные воздействия преобразуются в выходные результаты.

*Операция, работа.* Элемент работ проекта. У операций обычно имеется ожидаемая длительность, потребность в ресурсах, стоимость. Операции могут далее подразделяться на задачи.

В этих определениях существенным является следующее: состав, количество и, можно добавить, порядок выполнения фаз определяются особенностью проекта, а каждая фаза завершается получением одного из основных результатов, в то время как процесс или задача — просто значимого результата.

Для схемы модели жизненного цикла ПО характерно следующее:

- Результатом выполнения каждой фазы является некоторая модель ПО. Описание требований — модель того, что должен делать программный продукт; результат анализа — модель основных архитектурных решений и GUI.
- Результат выполнения каждой фазы является входом следующей фазы, и фазы должны выполняться в определенной моделью ЖЦ последовательности.
- Некоторые процессы могут выполняться на нескольких фазах, некоторые — в пределах одной.

В стандарте ISO 12207 модель жизненного цикла (life cycle model) определяется как структура, состоящая из процессов, работ и задач, включающих в себя разработку, эксплуатацию и сопровождение программного продукта, охватывающая жизнь системы от установления требований к ней до прекращения ее использования. При этом конкретные модели определяются особенностью задач, ограничениями на ресурсы, опытом разработчиков и т. д. Между тем, известны некоторые типовые модели ЖЦ ПО, которые проявили себя в определенных условиях, имеют определенные преимущества, недостатки и условия применимости. Эти типовые модели устанавливают некоторые принципы организации модели жизненного цикла ПО.

## Контрольные вопросы по главе 6

1) В чем сосит смысл жизненного цикла?
2) Укажите основные этапы общей модели жизненного цикла
3) С каким количеством классов ПО связан жизенный цикл?
4) Роль жизненного цикла для малых и больших ПО?
5) Опишите суть стандарта IEEE 1074 — процессы жизненного цикла для развития программного обеспечения.

6) Опишите суть стандарта ISO 12207 (15504) — Жизненный цикл ПП: структура и организация.
7) Укажите пять категорий процессов согласно стандарту ISO/IEC TR 15504.
8) Укажите группы инженерного процесса согласно стандарту ISO/IEC TR 15504.
9) Опишите модель жизненного цикла ПО согласно PMI.
10) Что характерно для схемы модели жизненного цикла ПО PMI?

# Глава 7

# КАСКАДНАЯ МОДЕЛЬ

Каскадная модель (от слова водопад — *waterfall*) впервые четко сформулирована в 1970 году Ройсом (W. W. Royce. Managing the Development of Large Software Systems http://facweb.cti.depaul.edu/). Она включает выполнение следующих фаз:

- *исследование концепции* — происходит исследование требований, разрабатывается видение продукта и оценивается возможность его реализации;
- *определение требований* — определяются программные требования для информационной предметной области системы, предназначение, линии поведения, производительность и интерфейсы;
- *разработка проекта* — разрабатывается и формулируется логически последовательная техническая характеристика программной системы, включая структуры данных, архитектуру ПО, интерфейсные представления и процессуальную (алгоритмическую) детализацию;
- *реализация* — эскизное описание ПО превращается в полноценный программный продукт. Результат: исходный код, база данных и документация. В реализации обычно выделяют два этапа: реализацию компонент ПО и интеграцию компонент в готовый продукт. На обоих этапах выполняются кодирование и тестирование, которые тоже иногда рассматривают как два подэтапа;
- *эксплуатация и поддержка* — подразумевает запуск и текущее обеспечение, включая предоставление технической помощи, обсуждение возникших вопросов с пользователем, регистрацию запросов пользователя на модернизацию и внесение изменений, а также корректирование или устранение ошибок;
- *сопровождение*- устранение программных ошибок, неисправностей, сбоев, модернизация и внесение изменений. Состоит из итераций разработки.

Основными принципами каскадной модели являются:

- Строго последовательное выполнение фаз:
  - Каждая последующая фаза начинается лишь тогда, когда полностью завершено выполнение предыдущей фазы.
  - Каждая фаза имеет определенные критерии входа и выхода: входные и выходные данные.
  - Каждая фаза полностью документируется.
  - Переход от одной фазы к другой осуществляется посредством формального обзора с участием заказчика.

На рис. 7.1 схематически изображена каскадная модель разработки программного обеспчения (ромбы соответствуют задачам, а стрелки — фазам). Внутри блока обозначена задача, которая должна быть решена на данном этапе разработки ПО. Связь между этапами показана только сверху вниз, тогда как в реальных процессах жизненного цикла следует учитывать возможность возврата на предшествующие этапы, снизу вверх, для их уточнения и корректировки результатов.


Рис. 7.1 – Каскадная модель разработки ПО

Основа модели — сформулированные требования (ТЗ), которые меняться не должны. Критерий качества результата — соответствие продукта установленным требованиям.

Каскадная модель имеет следующие преимущества:

- Проста и понятна заказчикам, т. к часто используется другими организациями для отслеживания проектов, не связанных с разработкой ПО.

- Проста и удобна в применении:
  - процесс разработки выполняется поэтапно;
  - ее структурой может руководствоваться даже слабо подготовленный в техническом плане или неопытный персонал;
  - она способствует осуществлению строгого контроля менеджмента проекта.
- Каждую стадию могут выполнять независимые команды (все документировано).
- Позволяет достаточно точно планировать сроки и затраты.

При использовании каскадной модели для «неподходящего» проекта могут проявляться следующие ее основные недостатки:

- попытка вернуться на одну или две фазы назад, чтобы исправить какую-либо проблему или недостаток, приведет к значительному увеличению затрат и сбою в графике;
- интеграция компонент, на которой обычно выявляется большая часть ошибок, выполняется в конце разработки, что сильно увеличивает стоимость устранения ошибок;
- запаздывание с получением результатов — если в процессе выполнения проекта требования изменились, то получится устаревший результат.

Недостатки каскадной модели особо остро проявляются в случае, когда трудно (или невозможно) сформулировать требования или требования могут меняться в процессе выполнения проекта.

В этом случае разработка ПО имеет принципиально циклический характер.

Методы и процессы стандартизации жизненного цикла ПС играют *стабилизирующую и организующую роль* во всем жизненном цикле многих сложных систем. Они обеспечивают:

- расширение и совершенствование функций систем и компонентов с сохранением их целостности и первичных затрат;
- систематическое повышение качества функционирования комплексов программ и баз данных для решения задач пользователей в различной внешней среде;
- улучшение технико-экономических характеристик применения систем и программных продуктов;
- совершенствование технологий обеспечения жизненного цикла сложных систем и комплексов программ.

Для этого при создании и сопровождении сложных, распределенных систем, формировании их архитектуры, при выборе стандартов для программных компонентов и их связей целесообразно учитывать ряд современных концептуальных требований программной инженерии и формирования их жизненного цикла:

- архитектура комплекса программ должна соответствовать текущим и перспективным целям и стратегическим, функциональным задачам создаваемой системы, быть достаточно гибкой и допускать относительно простое, без коренных структурных изменений, развитие и наращивание функций и ресурсов системы в соответствии с расширением сфер и задач ее применения;
- в структуре и компонентах ПС и системы следует предусматривать обеспечение максимально возможной сохранности инвестиций в аппаратные и программные средства, а также в базы данных при длительном развитии, сопровождении и модернизации системы;
- необходимо обеспечивать эффективное использование ресурсов в ЖЦ системы и минимизировать интегральные затраты на обработку данных в типовых режимах ее функционирования с учетом эксплуатационных затрат и капитальных вложений в создание системы и программного продукта;
- должны быть обеспечены безопасность функционирования системы и надежная защита данных от ошибок, от разрушения или потери информации, а также авторизация пользователей, управление рабочей загрузкой, резервированием и оперативным восстановлением функционирования системы и программного продукта;
- для обеспечения перспективы развития жизненного цикла системы и комплекса программ целесообразно предусматривать возможность интеграции гетерогенных вычислительных компонентов и возможность переноса ПС и БД на различные аппаратные и операционные платформы на основе концепции и стандартов открытых систем;
- следует обеспечить комфортное обучение и максимально упрощенный доступ конечных пользователей к управлению и результатам функционирования системы и программного продукта на основе современных графических средств и наглядных пользовательских интерфейсов.

Наиболее актуальна *стандартизация процессов жизненного цикла* комплексов программ при коллективной разработке и сопровождении крупных *критических систем управления в реальном времени*, к которым предъявляются высокие требования к качеству. В этих случаях особенно необходимо четкое планирование и управление технологическими процессами их жизненного цикла. Созданы или разрабатываются комплексы международных стандартов, в той или иной степени регламентирующие процессы проектирования, разработки, эксплуатации и сопровождения в ЖЦ программ и баз данных. Они обычно ориентированы на ПС, выполняющие важные функции в системах управления объектами, технологическими процессами или при обработке ответственной информации. Применение таких стандартов полностью при создании и использовании простых программ, узкого или экспериментального назначения (первого класса — см. выше) не всегда может быть оправдано. Однако они определяют современную культуру программной инженерии и стандартизации жизненного цикла комплексов программ высокого качества.

## Контрольные вопросы по главе 7

1) Каковы основные фазы каскадной модели жизненного цикла?
2) В чем сущность фазы «исследование концепции» и фазы определение требований»?
3) В чем сущность фазы «разработка проекта» и фазы «реализация»?
4) В чем состоит фаза «эксплуатация и поддержка» и «сопровождение»?
5) Что является основными принципами каскадной модели?
6) Схематически отобразить каскадную модель.
7) Укажите преимущества каскадной модели.
8) Укажите недостатки каскадной модели.
9) Что обеспечивает стандартизация жизненного цикла?
10) Какие требования целесообразно учитывать при выборе стандартов для программных компонентов и их связей?

# Глава 8

# СПИРАЛЬНАЯ МОДЕЛЬ

На практике, при решении достаточно большого количества задач, разработка ПО имеет циклический характер, когда после выполнения некоторых стадий приходится возвращаться на предыдущие. Можно указать две основные причины таких возвратов:

- Ошибки разработчиков, допущенные на ранних стадиях и выявленные на поздних стадиях — ошибки анализа, проектирования, кодирования, выявляемые, как правило, на стадии тестирования.
- Изменение требований в процессе разработки («ошибки» заказчиков). Это или неготовность заказчиков сформулировать требования («Сказать, что должна делать программа, я смогу только после того, как увижу как она работает»), или изменения требований, вызванные изменениями ситуации в процессе разработки (изменения рынка, новые технологии, . . .).

Спиральная модель была предложена Б. Боэмом в 1988 году (Barry Boehm, «A Spiral Model of Software Development and Enhancement», IEEE Computer, Vol.21, No.5, pp. 61–72, 1988. http://www.computer.org/computer/homepage/misc/Boehm) как альтернатива каскадной модели, учитывающая повторяющийся характер разработки ПО.

Основные принципы спиральной модели можно сформулировать следующим образом:

- Разработка вариантов продукта, соответствующих различным вариантам требований с возможностью вернуться к более ранним вариантам.
- Создание прототипов ПО как средства общения с заказчиком для уточнения и выявления требований.

- Планирование следующих вариантов с оценкой альтернатив и анализом рисков, связанных с переходом к следующему варианту.
- Переход к разработке следующего варианта до завершения предыдущего в случае, когда риск завершения очередного варианта (прототипа) становится неоправданно высок.
- Использование каскадной модели как схемы разработки очередного варианта.
- Активное привлечение заказчика к работе над проектом. Заказчик участвует в оценке очередного прототипа ПО, уточнении требований при переходе к следующему, оценке предложенных альтернатив очередного варианта и оценке рисков.

На рис. 8.1 схематически изображена спиральная модель разработки программного обеспечения (ромбы соответствуют задачам, а стрелки — фазам).

Спиральная модель. Схема работы спиральной модели выглядит следующим образом. Разработка вариантов продукта представляется как набор циклов раскручивающейся спирали. Каждому циклу спирали соответствует такое же количество стадий, как и в модели каскадного процесса. При этом начальные стадии, связанные с анализом и планированием, представлены более подробно с добавлением новых элементов.

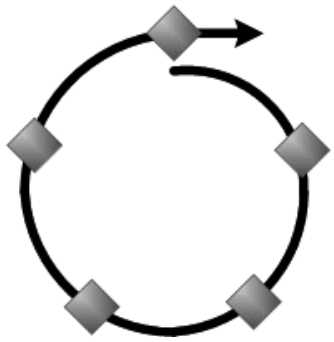


Рис. 8.1 – Спиральная модель разработки ПО

В каждом цикле выделяются четыре базовые фазы:

- определение целей, альтернативных вариантов и ограничений;
- оценка альтернативных вариантов, идентификация и разрешение рисков;
- разработка продукта следующего уровня;
- планирование следующей фазы.

«Раскручивание» проекта начинается с анализа общей постановки задачи на разработку ПО. Здесь на первой фазе определяются общие цели, устанавливаются предварительные ограничения, определяются возможные альтернативы подходов к решению задачи. Далее проводится оценка подходов, устанавливаются их риски. На шаге разработки создается концепция (видение) продукта и путей его создания.

Следующий цикл начинается с планирования требований и деталей ЖЦ продукта для оценки затрат. На фазе определения целей устанавливаются альтернативные варианты требований, связанные с аранжировкой требований по важности и стоимости их выполнения. На фазе оценки устанавливаются риски вариантов

требований. На фазе разработки — спецификация требований (с указанием рисков и стоимости), готовится демо-версия ПО для анализа требований заказчиком.

Последующий цикл — разработка проекта — начинается с планирования разработки. На фазе определения целей устанавливаются ограничения проекта (по срокам, объему финансирования, ресурсам, ...), определяются альтернативы проектирования, связанные с альтернативами требований, применяемыми технологиями проектирования, привлечением субподрядчиков. На фазе оценки альтернатив устанавливаются риски вариантов и делается выбор варианта для дальнейшей реализации. На фазе разработки выполняется проектирование и создается демо-версия, отражающая основные проектные решения.

Следующий цикл — реализация ПО — также начинается с планирования, основанного на знаниях, полученных на предыдущих циклах. Альтернативными вариантами реализации могут быть применяемые технологии реализации, привлекаемые ресурсы. Оценка альтернатив и связанных с ними рисков на этом цикле определяется степенью «отработанности» технологий и «качеством» имеющихся ресурсов. Фаза разработки выполняется по каскадной модели с выходом — действующим вариантном (прототипом) продукта.

Спиральная модель (по отношению к каскадной) имеет следующие очевидные преимущества:

- Более тщательное проектирование (несколько начальных итераций) с оценкой результатов проектирования, что позволяет выявить ошибки проектирования на более ранних стадиях.
- Поэтапное уточнение требований в процессе выполнения итераций, что позволяет более точно удовлетворить требованиям заказчика.
- Участие заказчика в выполнении проекта с использованием прототипов программы. Заказчик видит, что и как создается, не выдвигает необоснованных требований, оценивает реальные объемы финансирования.
- Планирование и управление рисками при переходе на следующие итерации позволяет разумно планировать использование ресурсов и обосновывать финансирование работ.
- Возможность разработки сложного проекта «по частям», выделяя на первых этапах наиболее значимые требования.

Основные недостатки спиральной модели связаны с ее сложностью:

- Сложность анализа и оценки рисков при выборе вариантов.
- Сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий).
- Сложность оценки точки перехода на следующий цикл.
- Бесконечность модели — на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки.

Спиральную модель целесообразно применять при следующих условиях:

- Когда пользователи не уверены в своих потребностях или когда требования слишком сложны и могут меняться в процессе выполнения проекта и необходимо прототипирование для анализа и оценки требований.
- Когда достижение успеха не гарантировано и необходима оценка рисков продолжения проекта.
- Когда проект является сложным, дорогостоящим и обоснование его финансирования возможно только в процессе его выполнения.
- Когда речь идет о применении новых технологий, что связано с риском их освоения и достижения ожидаемого результата.
- При выполнении очень больших проектов, которые в силу ограниченности ресурсов можно делать только по частям.

## Контрольные вопросы по главе 8

1) Каковы основные принципы спиральной модели?
2) Сколько циклов спирали возникает при разработке ПО?
3) Сколько есть базовых фаз в каждом цикле спирали?
4) Какие задачи ставятся на первой фазе разработки ПО в спиральной модели?
5) Что устанавливается на фазе определения целей в спиральной модели?
6) С какого элемента разработки начинается разработка проекта в спиральной модели?
7) Укажите преимущества и недостатки спиральной модели разработки ПО?
8) В каких условиях выгодно применять спиральную модель?

# Глава 9

# ДРУГИЕ ТИПЫ МОДЕЛЕЙ ЖЦ ПО

Каскадная и спиральная модели устанавливают некоторые принципы организации жизненного цикла создания программного продукта. Каждая из них имеет преимущества, недостатки и области применимости. Каскадная модель проста, но применима в случае, когда требования известны и меняться не будут. Спиральная модель учитывает такие важные показатели проекта, как изменяемость требований, невозможность оценить заранее объем финансирования, риски выполнения проекта. Но спиральная модель сложна и требует больших затрат на сопровождение. Существуют некоторые другие типы моделей, которое можно рассматривать как «промежуточные» между каскадной и спиральной моделями. Эти модели используют отдельные преимущества каскадной и спиральной моделей и достигают успеха для определенных типов задач.

## Итерационная модель

Итерационная модель жизненного цикла является развитием классической каскадной модели и предполагает возможность возвратов на ранее выполненные этапы. При этом причинами возвратов в классической итерационной модели являются выявленные ошибки, устранение которых и требует возврата на предыдущие этапы в зависимости от типа (природы) ошибки — ошибки кодирования, проектирования, спецификации или определения требований. Реально итерационная модель является более жизненной, чем классическая (строгая) каскадная модель, т. к. создание ПО всегда связано с устранением ошибок.

## V-образная модель

V-образная модель была создана как итерационная разновидность каскадной модели. Целями итераций в этой модели является обеспечение процесса тестирова-

ния. Тестирование продукта обсуждается, проектируется и планируется на ранних этапах жизненного цикла разработки. План испытания приемки заказчиком разрабатывается на этапе планирования, а компоновочного испытания системы — на фазах анализа, разработки проекта и т. д. Этот процесс разработки планов испытания обозначен пунктирной линией между прямоугольниками V-образной модели. Помимо планов, на ранних этапах разрабатываются также и тесты, которые будут выполняться при завершении параллельных этапов.

## Инкрементная (пошаговая) модель

Инкрементная разработка представляет собой процесс поэтапной реализации всей системы и поэтапного наращивания (приращения) функциональных возможностей. На первом шаге необходим полный заранее сформулированный набор требований, которые делятся по некоторому признаку на части. Далее выбирается первая группа требований и выполняется полный проход по каскадной модели. После того, как первый вариант системы, выполняющий первую группу требований, сдан заказчику, разработчики переходят к следующему шагу (второму инкременту) по разработке варианта, выполняющего вторую группу требований.

Особенностью инкрементной модели является разработка приемочных тестов на этапе анализа требований, что упрощает приемку варианта заказчиком и устанавливает четкие цели разработки очередного варианта системы. Инкрементная модель особенно эффективна в случае, когда задача разбивается на несколько относительно независимых подзадач (разработка подсистем «Зарплата», «Бухгалтерия», «Склад», «Поставщики»). Кроме того, инкрементная модель может для внутренней итерации использовать не только каскадную, но и другие типы моделей.

## Модель быстрого прототипирования

Модель быстрого протитипирования предназначена для быстрого создания прототипов продукта с целью уточнения требований и поэтапного развития прототипов в конечный продукт. Скорость (высокая производительность) выполнения проекта обеспечивается планированием разработки прототипов и участием заказчика в процессе разработки. Начало жизненного цикла разработки помещено в центре эллипса. Совместно с пользователем разрабатывается предварительный план проекта на основе предварительных требований. Результат начального планирования — документ, описывающий в общих чертах примерные графики и результативные данные.

Следующий уровень — создание исходного прототипа на основе быстрого анализа, проекта баз данных, пользовательского интерфейса и некоторых функций. Затем начинается итерационный цикл быстрого прототипирования. Разработчик проекта демонстрирует очередной прототип, пользователь оценивает его функционирование, совместно определяются проблемы и пути их преодоления для перехода к следующему прототипу. Этот процесс продолжается до тех пор, пока пользователь не согласится, что очередной прототип в точности отображает все требования.

Получив одобрение пользователя, быстрый прототип преобразуют в детальный проект и систему настраивают на производственное использование. Именно на этом этапе настройки ускоренный прототип становится полностью действующей системой.

При разработке производственной версии программы может понадобиться более высокий уровень функциональных возможностей, различные системные ресурсы, необходимые для обеспечения полной рабочей нагрузки, или ограничения во времени. После этого следуют тестирование в предельных режимах, определение измерительных критериев и настройка, а затем, как обычно, функциональное сопровождение.

## Модели жизненного цикла MSF, RUP, XP

В настоящее время широкое применение получают так называемые промышленные технологии создания программного продукта. Эти технологии были разработаны фирмами, накопившими большой опыт создания ПО. Технологии представлены описаниями принципов, методов, применяемых процессов и операций. Такие технологии, как правило, поддерживаются набором CASE-средств, охватывают все этапы жизненного цикла продукта и успешно применяются для решения практических задач.

Хороший обзор моделей жизненного цикла промышленных технологий можно прочитать в работе Скопина И.Н. «Основы менеджмента программных проектов. Лекция № 11: Модели жизненного цикла в некоторых реальных методологиях программирования»: http://www.intuit.ru/department/se/msd/11/1.html. Здесь мы рассмотрим особенности моделей жизненного цикла трех наиболее известных промышленных технологий:

## Microsoft Solution Framework (MSF)

Разработка фирмы Microsoft, предназначенная для решения широкого круга задач. Технология масштабируема, т. е. настраиваема на решение задач любой сложности коллективом любой численности.

Одна из особенностей технологии MSF состоит в том, что она ориентирована не просто на создание программного продукта, удовлетворяющего перечисленным требованиям, а на поиск решения проблем, стоящих перед заказчиком.

Различие состоит в том, что перечисляемые заказчиком требования являются проявлениями некоторых более глубоких проблем и неточность, неполнота, изменение требований в процессе разработки — следствие недопонимания проблем. Поэтому в технологии MSF большое внимание уделяется анализу проблем заказчика и разработке вариантов системы для поиска решения этих проблем.

Модель жизненного цикла MSF является некоторым гибридом каскадной и спиральной моделей, сочетая простоту управления каскадной модели с гибкостью спиральной.

Модель жизненного цикла MSF ориентирована на «вехи» (*milestones*) — ключевые точки проекта, характеризующие достижение какого-либо существенного результата. Этот результат может быть оценен и проанализирован, что подразумевает ответ на вопрос: «А достигли ли мы целей, поставленных на этом шаге?». В модели предусматривается наличие основных вех (завершение главных фаз модели) и промежуточных, отражающих внутренние этапы главных фаз.

Основными фазами модели MSF являются:

*Создание общей картины приложения (Envisioning).* На этом этапе решаются следующие основные задачи: оценка существующей ситуации; определение состава команды, структуры проекта, бизнес-целей, требований и профилей пользователей; разработка концепции решения и оценка риска. Устанавливаются две промежуточные вехи: «Организован костяк команды» и «Создана общая картина решения».

*Планирование (Panning).* Включает планирование и проектирование продукта. На основе анализа требований разрабатывается проект и основные архитектурные решения, функциональные спецификации системы, планы и календарные графики, среды разработки, тестирования и пилотной эксплуатации. Этап состоит из трех стадий: концептуальное, логическое и физическое проектирование. На стадии *концептуального* проектирования задача рассматривается с точки зрения пользовательских и бизнес-требований и заканчивается определением набора сценариев использования системы. При *логическом* проектировании задача рассматривается с точки зрения проектной команды, решение представляется в виде набора сервисов. И уже на стадии *физического* проектирования задача рассматривается с точки зрения программистов, уточняются используемые технологии и интерфейсы.

*Разработка (Developing).* Создается вариант решения проблемы в виде кода и документации очередного прототипа, включая спецификации и сценарии тестирования. Основная веха этапа — «Окончательное утверждение области действия проекта». Продукт готов к внешнему тестированию и стабилизации. Кроме того, заказчики, пользователи, сотрудники службы поддержки и сопровождения, а также ключевые участники проекта могут предварительно оценить продукт и указать все недостатки, которые нужно устранить до его поставки.

*Стабилизация (Stabilizing).* Подготовка к выпуску окончательной версии продукта, доводка его до заданного уровня качества. Здесь выполняется комплекс работ по тестированию (обнаружение и устранение дефектов), проверяется сценарий развертывания продукта. Когда решение становится достаточно устойчивым, проводится его пилотная эксплуатация в тестовой среде с привлечением пользователей и применением реальных сценариев работы.

*Развертывание (Deploying).* Выполняется установка решения и необходимых компонентов окружения, проводится его стабилизация в промышленных условиях и передача проекта в руки группы сопровождения. Кроме того, анализируется проект в целом на предмет уровня удовлетворенности заказчика.

Подробнее модель процессов MSF, версии 3.1 можно найти на сайте разработчика: http://www.microsoft.com/Rus/.

# Rational Unified Process (RUP)

Разработка фирмы Rational, долгое время успешно занимавшейся созданием CASE-средств, применяемых на различных этапах жизненного цикла продукта от анализа до тестирования и документирования. Аналогично MSF, RUP универсальна, масштабируема и настраиваема на применение в конкретных условиях.

Модель жизненного цикла RUP является довольно сложной, детально проработанной итеративно-инкрементной моделью с элементами каскадной модели. В модели RUP выделяются 4 основные фазы, 9 видов деятельности (процессов). Кроме того, в модели описывается ряд практик, которые следует применять или руководствоваться для успешного выполнения проекта. RUP ориентирован на поэтапное моделирование создаваемого продукта с помощью языка UML.

Основными фазами RUP являются:

- *Фаза начала проекта (Inception).* Определяются основные цели проекта, бюджет проекта, основные средства его выполнения — технологии, инструменты, ключевой персонал, составляются предварительные планы проекта. Основная цель этой фазы — достичь компромисса между всеми заинтересованными лицами относительно задач проекта.
- *Фаза проработки (Elaboration).* Основная цель этой фазы — на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы.
- *Фаза построения (Construction).* Основная цель этой фазы — детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры.
- *Фаза передачи (Transition).* Цель фазы — сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей.

В рамках каждой фазы возможно проведение нескольких итераций, количество которых определяется сложностью выполняемого проекта.

Деятельности (основные процессы) RUP делятся на пять рабочих и четыре поддерживающие. К рабочим деятельностям относятся:

*Моделирование предметной области (бизнес-моделирование, Business Modeling).* Цели этой деятельности — понять бизнес-контекст, в котором должна будет работать система (и убедиться, что все заинтересованные лица понимают его одинаково), понять возможные проблемы, оценить возможные их решения и их последствия для бизнеса организации, в которой будет работать система.

*Определение требований (Requirements).* Цели — понять, что должна делать система, определить границы системы и основу для планирования проекта и оценок ресурсозатрат в нем.

*Анализ и проектирование (Analysis and Design).* Выработка архитектуры системы на основе ключевых требований, создание проектной модели, представленной в виде диаграмм UML, описывающих продукт с различных точек зрения.

*Реализация (Implementation).* Разработка исходного кода, компонент системы, тестирование и интегрирование компонент.

*Тестирование (Test).* Общая оценка дефектов продукта, его качество в целом; оценка степени соответствия исходным требованиям.

Поддерживающими деятельностями являются:

*Развертывание (Deployment).* Цели — развернуть систему в ее рабочем окружении и оценить ее работоспособность.

*Управление конфигурациями и изменениями (Configuration and Change Management).* Определение элементов, подлежащих хранению, и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.

*Управление проектом (Project Management).* Включает планирование, управление персоналом, обеспечение связей с другими заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.

*Управление средой проекта (Environment).* Настройка процесса под конкретный проект, выбор и смена технологий и инструментов, используемых в проекте.

## RAD (Rapid Application Development)

Одним из возможных подходов к разработке программного обеспечения (ПО) в рамках спиральной модели жизненного цикла является методология быстрой разработки приложений RAD (Rapid Application Development).

Под этим термином обычно понимается процесс разработки ПО, содержащий три элемента:

- небольшую команду программистов (от 2 до 10 человек);
- короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);
- повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Команда разработчиков должна представлять из себя группу профессионалов, имеющих опыт в анализе, проектировании, генерации кода и тестировании ПО с использованием CASE-средств. Члены коллектива должны также уметь трансформировать в рабочие прототипы предложения конечных пользователей. Жизненный цикл ПО по методологии RAD состоит из четырех фаз:

- фаза анализа и планирования требований;
- фаза проектирования;
- фаза построения;
- фаза внедрения.

На фазе анализа и планирования требований пользователи системы определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Определение требований выполняется в основном силами пользо-

вателей под руководством специалистов-разработчиков. Ограничивается масштаб проекта, определяются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации данного проекта в установленных рамках финансирования, на данных аппаратных средствах и т. п. Результатом данной фазы должны быть список и приоритетность функций будущей ИС, предварительные функциональные и информационные модели ИС.

На фазе проектирования часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и, при необходимости, корректируется функциональная модель. Каждый процесс рассматривается детально. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Определяются требования разграничения доступа к данным. На этой же фазе происходит определение набора необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении ИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время — порядка 60–90 дней. С использованием CASE-средств проект распределяется между различными командами (делится функциональная модель). Результатом данной фазы должны быть:

- общая информационная модель системы;
- функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;
- точно определенные с помощью CASE-средства интерфейсы между автономно разрабатываемыми подсистемами;
- построенные прототипы экранов, отчетов, диалогов.

Все модели и прототипы должны быть получены с применением тех CASE-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что в традиционном подходе при передаче информации о проекте с этапа на этап может произойти фактически неконтролируемое искажение данных. Применение единой среды хранения информации о проекте позволяет избежать этой опасности.

В отличие от традиционного подхода, при котором использовались специфические средства прототипирования, не предназначенные для построения реальных приложений, а прототипы выбрасывались после того, как выполняли задачу устранения неясностей в проекте, в подходе RAD каждый прототип развивается в часть будущей системы. Таким образом, на следующую фазу передается более полная и полезная информация.

На фазе построения выполняется непосредственно сама быстрая разработка приложения. На данной фазе разработчики производят итеративное построение реальной системы на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формиру-

ется при помощи автоматических генераторов, получающих информацию непосредственно из репозитория CASE-средств. Конечные пользователи на этой фазе оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям.

Тестирование системы осуществляется непосредственно в процессе разработки.

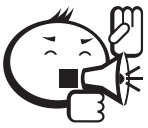
После окончания работ каждой отдельной командой разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения с остальными, а затем тестирование системы в целом. Завершается физическое проектирование системы:

- определяется необходимость распределения данных;
- производится анализ использования данных;
- производится физическое проектирование базы данных;
- определяются требования к аппаратным ресурсам;
- определяются способы увеличения производительности;
- завершается разработка документации проекта.

Результатом фазы является готовая система, удовлетворяющая всем согласованным требованиям. На фазе внедрения производятся обучение пользователей, организационные изменения и параллельно с внедрением новой системы осуществляется работа с существующей системой (до полного внедрения новой). Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на этапе проектирования системы. Приведенная схема разработки ИС не является абсолютной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется разработка: разрабатывается совершенно новая система; уже было проведено обследование предприятия и существует модель его деятельности; на предприятии уже существует некоторая ИС, которая может быть использована в качестве начального прототипа или должна быть интегрирована с разрабатываемой.

Следует, однако, отметить, что методология RAD, как и любая другая, не может претендовать на универсальность, она хороша в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика. Если же разрабатывается типовая система, которая не является законченным продуктом, а представляет собой комплекс типовых компонент, централизованно сопровождаемых, адаптируемых к программно-техническим платформам, СУБД, средствам телекоммуникации, организационно-экономическим особенностям объектов внедрения и интегрируемых с существующими разработками, на первый план выступают такие показатели проекта, как управляемость и качество, которые могут войти в противоречие с простотой и скоростью разработки. Для таких проектов необходимы высокий уровень планирования и жесткая дисциплина проектирования,

строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.



Методология RAD неприменима для построения сложных расчетных программ, операционных систем или программ управления космическими кораблями, т. е. программ, требующих написания большого объема (сотни тысяч строк) уникального кода.

Не подходят для разработки по методологии RAD приложения, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени) и приложения, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

Оценка размера приложений производится на основе так называемых функциональных элементов (экраны, сообщения, отчеты, файлы и т. п.). Подобная метрика не зависит от языка программирования, на котором ведется разработка. Размер команды разработчиков приложения, которое может быть выполнено по методологии RAD, можно определить следующим образом: 1000 функциональных элементов — один человек; 1000–4000 функциональных элементов — одна команда разработчиков (3–5 человек); 4000 функциональных элементов — 4000 функциональных элементов на каждую команду разработчиков (более 5 человек).

## Extreme Programming (XP)

Extreme Programming (XP) — активно развивающаяся в последнее время технология, предназначенная для решения относительно небольших задач относительно небольшими коллективами профессиональных разработчиков в условиях жестко ограниченного времени.

Модель жизненного цикла XP является итерационно-инкрементной моделью быстрого создания (и модификации) прототипов продукта, удовлетворяющих очередному требованию (user story). Особенности этой модели представлены на слайде. Основными фазами модели можно считать:

1) «Вброс» архитектуры — начальный этап проекта, на котором создается видение продукта, принимаются основные решения по архитектуре и применяемым технологиям. Результатом начального этапа является метафора (metaphor) системы, которая в достаточно простом и понятном команде виде должна описывать основной механизм работы системы.

2) Истории использования (User Story) — этап сбора требований, записываемых на специальных карточках в виде сценариев выполнения отдельных функций. Истории использования являются требованиями для планирования очередной версии и одновременной разработки приемочных тестов (Acceptance tests) для ее проверки.

3) Планирование версии (релиза). Проводится на собрании с участием заказчика путем выбора User Stories, которые войдут в следующую версию. Одновременно принимаются решения, связанные с реализацией версии. Цель планирования — получение оценок того, что и как можно сделать за 1–3 недели создания следующей версии продукта.
4) Разработка проводится в соответствии с планом и включает только те функции, которые были отобраны на этапе планирования.
5) Тестирование проводится с участием заказчика, который участвует в составлении тестов.
6) Выпуск релиза — разработанная версия передается заказчику для использования или бета-тестирования.

По завершению цикла делается переход на следующую итерацию разработки.

Особенности модели жизненного цикла XP проясняют следующие принципы этого метода. Прежде всего, это принципы «живой» разработки ПО, зафиксированные в манифесте «живой» разработки:

- Люди, их общение более важны, чем процессы и инструменты.
- Работающая программа более важна, чем исчерпывающая документация.
- Сотрудничество с заказчиком более важно, чем обсуждение деталей контракта.
- Отработка изменений более важна, чем следование планам.

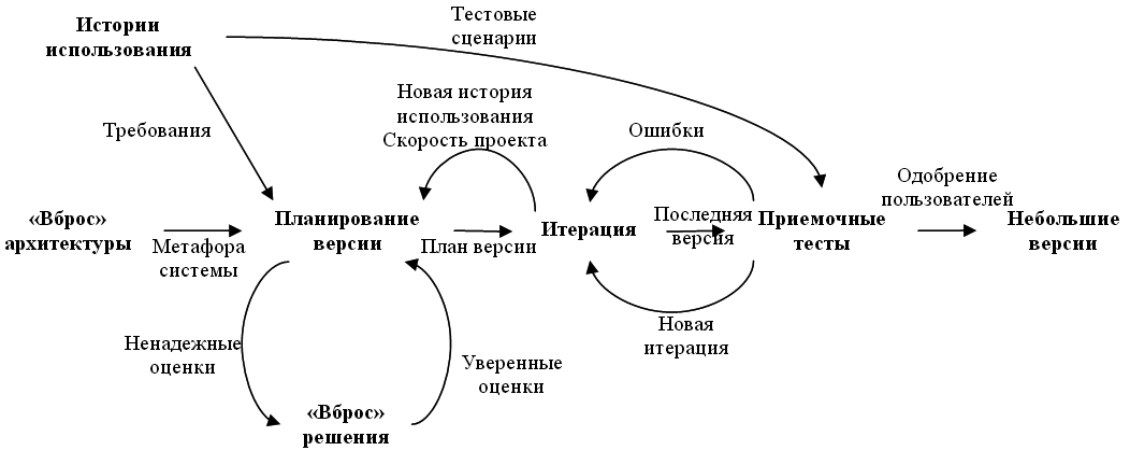



Рис. 9.1 – Схема процессов экстремального программирования (XP)

Кроме того, в XP есть несколько правил (техник), характеризующих особенности модели его жизненного цикла:

*Живое планирование (planning game)* — как можно быстрее определить объем работ, который нужно сделать до следующей версии ПО. Решение принимается на основе, в первую очередь, бизнес-приоритетов заказчика и, во вторую, технических оценок. Планы изменяются, как только они начинают расходится с действительностью или пожеланиями заказчика.

*Частая смена версий (small releases)* — первая работающая версия должна появиться как можно быстрее и тут же должна начать использоваться. Следующие версии подготавливаются через достаточно короткие промежутки времени.

*Простые проектные решения (simple design)* — в каждый момент времени система должна быть сконструирована так просто, насколько это возможно. Новые функции добавляются только после ясной просьбы об этом. Вся лишняя сложность удаляется, как только обнаруживается.

*Разработка на основе тестирования (test-driven development)* — сначала пишутся тесты, потом реализуются модули так, чтобы тесты срабатывали. Заказчики заранее пишут тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала.

*Постоянная переработка (refactoring)* — системы для устранения излишней сложности, увеличения понятности кода, повышения его гибкости. При этом предпочтение отдается более элегантным и гибким решениям, по сравнению с просто дающими нужный результат.

*Программирование парами (pair programming)* — весь код пишется двумя программистами на одном компьютере, что повышает его качество (отсутствие ошибок, понятность, читаемость,. . .).

*Постоянная интеграция (continuous integration)* — система собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда пара программистов оканчивает реализацию очередной функции.

Более подробно об экстремальном программировании можно почитать здесь: http://www.xprogramming.ru/index.html.

Классификацию современных моделей жизненного цикла ПО по типам проектов можно найти в обзоре: Рассел Арчибальд. Модели жизненного цикла высокотехнологичных проектов http://www.pmprofy.ru/content/rus/107/1073-article.asp.

## Контрольные вопросы по главе 9

1) В чем состоит итерационная модель жизненного цикла разработки ПО?
2) В чем состоит V-образная модель жизненного цикла разработки ПО?
3) В чем состоит пошаговая модель жизненного цикла разработки ПО?
4) В чем состоит модель быстрого прототипирования жизненного цикла разработки ПО?
5) В чем состоит модель MSF жизненного цикла разработки ПО?
6) В чем состоит модель RUP жизненного цикла разработки ПО?
7) В чем состоит модель RAD жизненного цикла разработки ПО?
8) В чем состоит модель XP экстримального программирования разработки ПО?
9) Приведите правила модели жизненного цикла XP?
10) Укажите основные отличия моделей жизненного цикла?

# Глава 10

# УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

В различных литературных источниках можно найти различные определения понятия «управление» [14]:

- Управление — элемент, функция организованных систем различной природы (биологических, социальных, технических), обеспечивающая сохранение их определенной структуры, поддержание режима деятельности, реализацию их программ и целей (Советская энцикопедия).
- Управление — руководство, направление чей-либо деятельности (http://www.mega.km.ru).
- Управление — изменение состояния объекта, системы или процесса, ведущее к достижению поставленной цели (словарь по кибернетике).

С точки зрения последнего (наиболее приемлемого для нас) определения, существенным является:

- наличие цели управления;
- наличие (возможность) управляющего воздействия;
- наличие измерений состояния объекта или процесса;
- ограниченность управления.

При программировании часто говорят так: «выполнить проект» (т. е. написать программу и документацию). Так что же такое проект? Слово «проект» имеет достаточно много значений. Происходит от латинского projectus, что означает «брошенный вперед». Под проектом обычно понимается некоторый достаточно сложный вид деятельности, управление которым является также достаточно сложным и при удаче может принести хороший результат. Известны несколько определений проекта:

- Проект — это произвольный ряд действий или задач, имеющий определенную цель, которая будет достигнута в рамках выполнения некоторых заданий, характеризующихся определенными датами начала и окончания, пределами финансирования и ресурсами (Г. Керцнер).
- Проект — одноразовая работа, которая имеет определенные даты начала и окончания, ясно определенные цели, возможности и, как правило, бюджет (Д. Льюис).
- Проект — временное усилие, применяемое для того, чтобы создать уникальный продукт или услугу с определенной датой начала и окончания действия, отличающегося от продолжающихся, повторных действий и требующего прогрессивного совершенствования характеристик (PMI).

В этих определениях в той или иной степени отражаются следующие существенные характеристики проекта:

1) Цель проекта. Наличие четко выраженного конечного результата, выхода, продукции, определяемых в терминах затрат, качества и времени реализации.
2) Уникальность. Проект — это разовое начинание, которое не будет повторяться. Даже «повторяющиеся» проекты, например по строительству еще одного предприятия по той же проектной документации, значительно отличаются друг от друга использующимися ресурсами и средой реализации.
3) Ограниченность во времени. Проект имеет начало и конец. Для его реализации необходима временная концентрация ресурсов. По минованию надобности ресурсы используются на другие цели.
4) Ограниченность ресурсов, выделяемых на выполнение проекта (финансовых, людских, материальных).
5) Сложность. Для достижения целей проекта необходимо решить множество задач. Отношения между задачами могут быть довольно сложными, особенно, если в проекте много задач.
6) Неопределенность. Возможность достижения цели в указанные сроки с выделенными ресурсами заранее не гарантирована.
7) Предсказуемость. По мере реализации проекта изменяется потребность в тех или иных ресурсах. Это изменение идет в определенной предсказуемой последовательности, определяемой жизненным циклом проекта.

Иными словами, проект — это достаточно сложный вид деятельности, которым сложно управлять в силу его уникальности и ограниченности ресурсов и времени. Это обстоятельство вносит в проект элемент неопределенности, а правильно организованное управление делает результаты предсказуемыми. Кстати, предсказуемый — не значит успешный. Это значит — во время завершенный (успешно) или во время прекращенный (неуспешно).

Если есть проект, тогда для его выполнения необходимо вводить управление. Известны несколько определений управления проектами:

- Набор проверенных принципов, методов и методик, применяемых для эффективного планирования, составления графика, управления и отслеживания результатов работы (PMI).

- Планирование, организация, контроль и управление ресурсами компании, выделенными в рамках определенного проекта (Керцнер — Kerzner).
- Специализация общего менеджмента, определяющего применение персоналом стандартных руководящих навыков планирования, организации, комплектования, продвижения, а также управления и контроля для достижения определенных целей проекта.

*Наиболее полным можно считать определение, сформулированное PMI в Своде знаний по управлению проектами (PMBOK): «**Управление проектом (Project Management — PM)** — это наука и искусство руководства и координации людских и материальных ресурсов на протяжении жизненного цикла проекта путем применения современных методов и техники управления для достижения определенных в проекте результатов по составу и объему работ, стоимости, времени, качеству и удовлетворению участников проекта».*

Управление проектом основано на двух принципах:

1) Умение — знание принципов и методов управления проектом (планирования, организация, составление графиков, контроль, управление и отслеживание).
2) Навыки — опыт в области управления — применение умения для достижения целей в конкретных условиях.

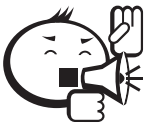
Хотя разработка методов и приемов управления была начата еще в начале прошлого века, дисциплина «управление проектами» начала складываться в 50-х годах XX столетия, что было вызвано необходимостью координации работ в крупных проектах по разработке вооружений и освоению космоса (США). Разрабатывались методы управления крупными проектами, среди которых наиболее известными являются:

1) Метод критического пути — *МКП* (CPM — Critical Path Method).
2) Метод анализа и оценки программ *PERT* (Program Evaluation and Review Technique).

В настоящее время в США почти не осталось компаний, которые не используют формальные методы управления проектами. В России формальные методы в проектах использует незначительное число предприятий. Большинство из этих инноваторов работают на рынке информационных технологий. Согласно исследованию, проведенному консалтинговой компанией Interthink, 97,5% компаний в США и Канаде используют формализованные подходы к управлению проектами, а 22,5% компаний используют полностью проектно-ориентированный подход для всех своих проектов (http://www.pmprofy.ru/content/rus/89/894-article.asp).

В России же ситуация прямо противоположная. По оценкам экспертов, только 5% (http://www.mdi.ru/aspnews/body/01.08.2002_61638.html) компаний используют те или иные формальные подходы к управлению проектами. Между тем, в России

также наблюдается взрывообразный интерес к методам и стандартам управления проектами. Любопытно, что большая часть из этих 5% российских предприятий приходится на IT-компании.



Управление проектом должно придерживаться определенного направления, которое определяется категориями.

Категории (от греч. *kategoria* высказывание; признак) в философии — наиболее общие и фундаментальные понятия, отражающие существенные всеобщие свойства и отношения явлений действительности и познания. Категории образовались как результат обобщения исторического развития познания и практики: материя и сознание, пространство и время, причинность, необходимость и случайность, возможность и действительность и др.

Аналогично философским категориям, в области управления проектами существуют категории, отражающие основные понятия этой области. В общем случае выделяют следующие группы категорий:

- *Цели*, определяемые ожидаемыми результатами проекта.
- *Критерии успеха и ограничения*: стоимость, сроки, качество.
- *Основные рычаги*: ресурсы (являющиеся также ограничением) и технологии.
- *Вспомогательные рычаги*: контракты, организация, взаимодействие, персонал.
- *Неопределенность*, связанная с рисками выполнения проекта.

Ключевой категорией, участвующей в процессе управления проектами, являются ограничения. Известный закон Лермана гласит: «Любую техническую проблему можно преодолеть, имея достаточно времени и денег», а следствие Лермана уточняет: «Вам никогда не будет хватать либо времени, либо денег». Именно эти три момента: время, бюджет и качество работ находятся под постоянным вниманием руководителя проекта. Их также можно назвать основными ограничениями, накладываемыми на проект.

Указанные выше три ограничения (сроки, расходы и качество результата) взаимосвязаны. Для иллюстрации взаимосвязи используют треугольник ограничений, в котором качество, время и деньги интерпретируются площадями внутренних треугольников. В этом треугольнике центр и верхняя вершина фиксированы, а нижние вершины могут перемещаться. Треугольник иллюстрирует, что любое сокращение финансов или времени ведет к сокращению качества, а увеличение качества может быть достигнуто за счет увеличения финансирования или сроков.

Набор действий, которые надо знать, чтобы управление было качественным определятся в PMBOK (Project Management Body of Knowledge — Свод знаний по управлению проектами) — международный стандарт состава знаний по управлению проектами, который разработан и развивается Институтом Проектного Менеджмента (Project Management Institute — PMI). Например, в версии стандарта 2004 г. содержатся описания состава знаний по следующим 9 разделам (областям знаний) управления проектами:

*1. Управление интеграцией проекта (Integration):*

- Создание плана проекта (Project Plan Development).
- Исполнение плана проекта (Project Plan Execution).
- Контроль изменений в проекте (Integrated Change Control).
- Управление объемом работ (Scope).
- Инициирование (Initiation) — формальное принятие решения о начале проекта (следующей фазы проекта).
- Планирование объема работ (Scope Planning) — разработка документа, описывающего объем работ.
- Формализация объема работ (Scope Definition) — декомпозиция всего объема работ (основных необходимых результатов) на мелкие, измеримые задачи.
- Верификация (Scope Verification) — подтверждение объема работ — формальная проверка приемлемости результатов работы.

*2. Управление изменениями объема работ (Scope Change Control) — контроль и утверждение изменений.*

*3. Управление временем выполнения (Time):*

- Определение состава работ (Activity Definition).
- Определение взаимосвязей работ (Activity Sequencing).
- Оценка длительностей работ (Activity Duration Estimating).
- Составление расписания проекта (Schedule Development).

*4. Управление стоимостью (Cost):*

- Планирование ресурсов (Resource Planning) — какие ресурсы, в каком количестве нужны для работ.
- Оценка стоимостей (Cost Estimating) — ресурсов, необходимых для работ.
- Разработка бюджета (Cost Budgeting) — бюджетирование — распределение затрат по компонентам проекта.
- Контроль стоимости (Cost Control) — управление изменениями бюджета.

*5. Управление качеством (Quality):*

- Планирование качества (Quality Planning) — определение стандартов качества и средств для их достижения.
- Обеспечение качества процесса (Quality Assurance) — плановая, регулярная оценка исполнения — проверка производственных процессов.
- Контроль качества результатов (Quality Control) — мониторинг результатов проекта, определение их соответствия стандартам, выявление и устранение причин несоответствия качества.

*6. Управление персоналом (Human Resource):*

- Организационное планирование (Organizational Planning) — идентификация, документирование и назначение проектных ролей, обязанностей и структуры отчетности.
- Подбор кадров (Staff Acquisition) — получение необходимых для проекта человеческих ресурсов, назначение персонала в команду проекта.
- Развитие команды проекта (Team Development) — повышение производительности труда: индивидуальной и команды в целом (улучшение взаимодействия).

*7. Управление коммуникациями (Communications):*

- Планирование взаимодействия (Communications Planning) — определение потребностей участников проекта в информации и планирование информационных потоков.
- Распределение информации (Information Distribution) — регулярное и своевременное обеспечение участников проекта необходимой информацией.
- Оценка исполнения (Performance Reporting) — сбор и распространение отчетности о текущем состоянии проекта, достигнутом прогрессе и ожидаемых результатах.
- Административное завершение (Administrative Closure) — создание, распространение (уничтожение) информации, необходимые для формального завершения проекта/фазы.

*8. Управление рисками (Risk):*

- Планирование управления рисками (Risk Management Planning).
- Идентификация рисков (Risk Identification).
- Качественный анализ рисков (Qualitative Risk Analysis).
- Количественный анализ рисков (Quantitative Risk Analysis).
- Планирование реагирования на риски (Risk Response Planning).
- Мониторинг и контроль рисков (Risk Monitoring and Control).

*9. Управление закупками и поставками (Procurement):*

- Планирование закупок (Procurement Planning) — определение, какие продукты и услуги нужны извне.
- Планирование предложений (Solicitation Planning) — документирование требований к продуктам и услугам от внешних поставщиков.
- Получение предложений (Solicitation).
- Выбор поставщиков (Source Selection).
- Управление контрактами (Contract Administration) — регулирование отношений с поставщиками.
- Завершение контрактов (Contract Closeout) — подтверждение выполнения, разрешение споров.

Главное действующее лицо проекта — менеджер. Институтом качества ПО (SQI — Software Quality Institute) разработан руководящей документ (Body of Knowledge)

для сертификации менеджеров программных проектов (SWPM — SoftWare Project Management). В этом документе содержится список 34-х компетенций, которыми должен обладать менеджер программного проекта.

Список разделен на три основные категории:

1) Методика разработки продукта.
2) Навыки управления проектами.
3) Навыки управления персоналом.

*Методика разработки продукта* включает в себя такие направления, как:

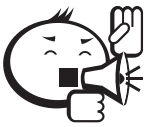
- Процессы оценивания — определение критериев для отбора.
- Знание стандартов процесса.
- Определение продукта — идентификация клиентской среды и требований, выдвигаемых к продукту.
- Оценка альтернативных процессов.
- Управление требованиями — мониторинг изменения требований.
- Управление субподрядчиками — планирование, управление и осуществление контроля.
- Выполнение начальной оценки — оценка степени трудности, рисков, затрат и создание графиков.
- Отбор методов и инструментов — определение процессов отбора.
- Подгонка процессов — модификация стандартных процессов с целью удовлетворения требований проектов.
- Отслеживание качества продукта — контроль качества в процессе разработки продукта.
- Понимание действий по разработке продукта — изучение цикла разработки ПО.

*Навыки управления проектами:*

- Создание структуры пооперационного перечня работ.
- Документирование планов — идентификация ключевых компонент.
- Оценка стоимости стоимости завершения проекта.
- Оценка трудозатрат необходимых для завершения проекта.
- Менеджмент рисков — идентификация, определение воздействия, обработка рисков.
- Отслеживание процесса разработки — контроль процесса разработки.
- Составление графика — разработка графика и ключевых стадий проекта.
- Выбор метрических показателей.
- Отбор инструментов менеджмента проекта — выбор методик и инструментов.
- Отслеживание процессов — мониторинг совместимости членов команды.
- Отслеживание хода разработки продукта — мониторинг хода разработки по выбранным метрическим показателям.

*Навыки управления персоналом:*

- Оценка производительности — оценка действий команды, направленных на повышение ее производительности.
- Вопросы интеллектуальной собственности — понимание степени влияния критических проблем.
- Организация эффективных встреч — планирование и проведение.
- Взаимодействие и общение — с разработчиками, руководством и другими командами.
- Лидерство — обучение проектных команд для получения оптимальных результатов.
- Управление изменениями — обеспечение эффективного управления изменениями.
- Успешное ведение переговоров — разрешение конфликтов и ведение переговоров.
- Планирование карьерного роста — структурирование и управление ходом реализации карьеры.
- Эффективное представление — использование письменных и устных навыков.
- Набор персонала — вербовка и собеседование с членами команды.
- Отбор команды высококомпетентных специалистов.
- Создание команды — формирование, руководство и поддержка эффективной команды.



Хотя все три задачи одинаково важны для успеха проекта, ведущую роль играет правильный подбор и управление командой.

Успех проекта во многом зависит от того, насколько состав участников проекта сможет быть преобразован в команду единомышленников, насколько эта команда будет активной и инициативной, с одной стороны, и управляемой, с другой. Из множества вопросов управления командой проекта мы рассмотрим три.

Состав команды определяется опытом и уровнем коллектива, особенностями проекта, применяемыми технологиями и уровнем этих технологий. В малых командах роли могут совмещаться. В больших — выделяться группы или отделы (отдел проектирования, отдел тестирования, отдел контроля качества, отдел подготовки документации и др.).

Состав команды определяется также типом выполняемых работ: под заказ или коробочное производство (продукт на рынок). В представленной модели можно выделить следующие основные роли:

*Менеджер проекта* — главное действующее лицо, обладающее знаниями и навыками, необходимыми для успешного управления проектом. Его основные функции:

- Подбор и управление кадрами.

- Подготовка и исполнение плана проекта.
- Руководство командой.
- Обеспечение связи между подразделениями.
- Обеспечение готовности продукта.

*Проектировщик* — это функция проектирования архитектуры высокого уровня и контроля ее выполнения. В небольших командах функция распределяется между менеджером и разработчиками. В больших проектах это может быть целый отдел. Основными функциями проектирования являются:

- Анализ требований.
- Разработка архитектуры и основных интерфейсов.
- Участие в планировании проекта.
- Контроль выполнения проекта.
- Участие в подборе кадров.

*Разработчик* — роль, ответственная за непосредственное создание конечного продукта. Помимо собственно программирования (кодирования), в его функции входит:

- Контроль архитектурных и технических спецификаций продукта.
- Подбор технологических инструментов и стандартов.
- Диагностика и разрешение всех технических проблем.
- Контроль за работой разработчиков документации, тестирования, технологов.
- Мониторинг состояния продукта (ведение списка обнаруженных ошибок).
- Подбор инструментов разработки, метрик и стандартов. Контроль их использования.

*Тестировщик* — роль, ответственная за удовлетворение требований к продукту (функциональных и нефункциональных). В функции тестировщика входит:

- Составление плана тестирования. План тестирования представляет один из элементов проекта и составляется до начала его реализации (разработки). Время, отводимое в плане на тестирование, может быть сопоставимо с временем разработки.
- Контроль выполнения плана. Важнейшая функция контроля — поддержка целостности базы данных зарегистрированных ошибок. В этой базе регистрируется:
  - кто, когда и где обнаружил, описание ошибки, описание состояния среды;
  - статус ошибки: приоритет, кто разрешает;
  - состояние ошибки: висит, в разработке, разрешена, проблемы.
- Эта база должна быть доступна всем, т. к. в тестировании принимают участие все члены команды.

- Разработка тестов. Самая трудоемкая часть в работе тестировщика. Тестирование должно обеспечить полную проверку функциональности при всех режимах работы продукта.
- Автоматизация тестирования включает автоматизацию составления тестов, автоматизацию пропуска тестов и автоматизацию обработки результатов тестирования. В виду важности автоматизации тестирования, иногда вводят нового участника — инженера по автоматизации.
- Выбор инструментов, метрик, стандартов для организации процесса тестирования.
- Организация Бета-тестирования — тестирования почти готового продукта внешними тестерами (пользователями). Эту важную процедуру надо продумать и организовать в случае разработки коробочного продукта.

*Инженер по качеству.* В современном представлении рассматривается три аспекта (уровня) качества:

- качество конечного продукта — обеспечивается тестированием,
- качество процесса разработки (тезис: для повышения качества продукта надо повысить качество процесса разработки),
- качество (уровень) организации (тезис: для повышения качества процесса надо повысить качество организации работ).

Повышение качества процессов требует участия всех действующих лиц. Принятое решение должно быть обосновано, всем понятно и всеми принято. При повышении качества организации работа по улучшению процессов проводится по определенной схеме. На каждом шаге повышения уровня организации работ выделяются ключевые процессы и выполняются работы по улучшению этих процессов.

*Технический писатель* или разработчик пользовательской (и иной) документации как части программного продукта. Функциями технического писателя являются:

- Разработка плана документирования, который включает состав, сроки подготовки и порядок тестирования документов. Выбор и разработка стандартов и шаблонов подготовки документов.
- Выбор средств автоматизации документирования.
- Разработка документации.
- Организация тестирования документации.
- Участие в тестировании продукта. Технический писатель все время работает с продуктом (его готовыми версиями) и, выступая от имени пользователя, видит все недочеты и несоответствия.

*Технолог разработки ПО* обеспечивает выполнение следующих задач:

- Поддержка модели ЖЦ — создание служб и структур по поддержке работоспособности принятой модели ЖЦ ПО. В поддержке модели ЖЦ принимают участие все. Но контроль возложен на технолога.
- Создание и сопровождение среды сборки продукта. Функция особенно важна на завершающих этапах разработки или при использовании модели прототипирования. В такой ситуации сборка будет проводиться достаточно

часто (в некоторых случаях — ежедневно). Среда сборки должна быть подготовлена заранее, сборка должна проводиться быстро и без сбоев. С учетом сборки версий это не простая задача.

- Создание и сопровождение процедуры установки с тем, чтобы каждая сборка устанавливалась автоматически с учетом версии и конфигураций сред.
- Управление исходными текстами — сопровождение и администрирование системы управления версиями исходных текстов.

## Контрольные вопросы по главе 10

1) Дайте определение понятия «управление», укажите существенные сущности понятия управления?
2) Дайте определение понятия характеристики проекта?
3) Укажите определения и принципы «управления проектами»?
4) Укажите категории и основные ограничения в «управлении проектом»?
5) Приведите набор дейсвий, которые надо знать, чтобы управление было качественным?
6) Кто является главным действующим лицом проекта?
7) Какие компетенции нужны для того, чтобы работать над проектом ПО?
8) Приведите основные разделы категории компетенций «Методика разработки продукта» и «Навыки управления проектами»?
9) Из каких соображений определяется состав команды исполнителей проекта?
10) Какие функции выполняет менеджер проекта, разработчик тестировщик, инженер по качеству, технический писатель и технолог разработки ПО?

# Глава 11

# МОДЕЛИ УПРАВЛЕНИЯ КОМАНДОЙ

***Управление*** *— это набор функций определенного круга лиц команды, выполняющих действия, способствующие выполнению проекта.*

Для этого необходимо управлять членами команды. Существует несколько моделей управления [20, 21].

## Административная модель (теория X)

Это традиционный стиль управления, связанный с иерархической административно-командной моделью, которую используют военные организации. В основе лежит теория X, которая утверждает, что такой подход необходим, поскольку большинство людей по своей природе не любит работу и будет стремиться избежать ее, если у них есть такая возможность. Однако менеджеры должны принуждать, контролировать, направлять сотрудников и угрожать им, чтобы получить от них максимальную отдачу. Девиз теории и модели: Люди делают только то, что вы контролируете. Или в более мягком варианте: Люди делают то, что они не хотят делать, только если вы их контролируете В конце концов, теория утверждает, что большинство людей предпочитают, чтобы им говорили, что следует делать и им не придется ничего решать самим.

Характерные черты модели:

- Властная пирамида — решения принимаются сверху-вниз.
- Четкое распределение ролей и обязанностей.
- Четкое распределение ответственности.
- Следование инструкциям, процедурам, технологиям.
- Роль менеджера: планирование, контроль, принятие основных решений.

Преимущества модели: ясность, простота, прогнозируемость. Модель хорошо сочетается с каскадной моделью жизненного цикла и применима в тех же случаях, что и каскадная модель. Модель эффективна в случае установившегося процесса. Недостатки модели связаны с тем, что административная система стремится к самосохранению (стабильности) и плохо восприимчива к изменению ситуации — новые типы проектов, применение новых технологий, оперативная реакция на изменение рынка. Кроме того, в административной модели плохо уживаются индивидуалисты и генераторы идей.

## Модель хаоса (теория Y)

В основе модели хаоса лежит Теория Y, которая является полной противоположностью Теории X. Основной тезис Теории Y: работа — естественная и приятная деятельность и большинство людей, на самом деле, очень ответственны и не увиливают от работы.

Характерными чертами модели хаоса являются:

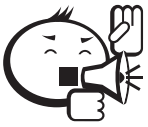
- Отсутствие явно выраженных признаков власти.
- Роль менеджера в постановке задачи, обеспечении ресурсами, не мешать и следить, чтобы не мешали другие.
- Отсутствие инструкций и регламентированных процедур.
- Применении индивидуальной инициативы.
- Решение проблем там, где они обнаружены.
- Процесс напоминает творческую игру участников на основе дружеской соревновательности.

Преимущества такой модели в том, что творческая инициатива участников ничем не связана и потенциал участников раскрывается в полной мере. Это бывает особенно эффективно в случае, когда для решения проблемы требуется поиск новых подходов, методов, идей и средств. Команда становится командой «прорыва», а работа проходит в форме игры, цель которой — поиск наилучшего результата. Процесс напоминает случайный поиск, когда идеи и решения рождаются при живом и как бы случайном обсуждении проблем в коридоре, столовой, на пикнике. Собрать такую команду в рабочей комнате и устроить обсуждение по регламенту часто просто не удается — это команда творческих индивидуалистов. Недостатки модели связаны с тем, что при определенных условиях команда прорыва может стать командой провала.

Творческая соревновательность переходит в конкуренцию сначала идей, а потом — личностей. Процесс начинает преобладать над целью проекта — высказанные идеи не доводятся до конца и сменяются новыми идеями, преобладание получают «красивые» идеи, лежащие в стороне от основных целей проекта. Люди, способные к генерации идей, редко обладают терпением доведения идей до полной реализации.

## Открытая архитектура (теория Z)

Административная и хаотическая модели являются двумя «крайностями», между которыми находятся множество моделей, сочетающих преимущества «крайних» моделей. Одной из таких моделей является модель открытой архитектуры, основанная на Теории Z. Эта теория была сформулирована Уильямом Оучи на основе изучения опыта японского стиля управления (Theory Z: How American Business Can Meet the Japanese Challenge. Perseus Publishing, 1981). Теория Z предполагает (но не декларирует) наличие внутреннего механизма управления, основанного на влиянии со стороны коллег и группы в целом. Дополнительное воздействие оказывают культурные нормы конкретной корпорации.



Основной принцип модели можно сформулировать так: «Работаем спокойно. Работаем вместе».

Особенностями этой модели являются:

- Адаптация к условиям работы — если делаем независимые модули, то расходимся и делаем, если нужна архитектура базы данных, то собираемся вместе и обсуждаем идеи.
- Коллективное обсуждение проблем, выработка консенсуса и принятие решения — не все могут согласиться, но принятое решение является коллективным и в силу этого — обязательным для всех.
- Распределенная ответственность — отвечают все, кто обсуждал, вырабатывал, принимал.
- Динамика состава рабочих групп в зависимости от текущих задач.
- Отсутствие специализации — участники меняются ролями и функциями и могут при необходимости заменить друг друга.
- Задача менеджера — активное (но рядовое, не руководящее) участие в процессе, контроль конструктивности обсуждений, обеспечение возможности активного участия всех.
- Открытая архитектура является более гибкой, адаптируемой, настраиваемой на ситуацию. Она дает возможность проявить себя всем членам команды — в ней могут уживаться и индивидуалисты, и коллективисты. Коллективное обсуждение высказанных идей позволяет оставлять только прагматичные идеи.

## Контрольные вопросы по главе 11

1) Приведите основные модели управления командой разработчиков ПО?
2) В чем состоит «Административная модель (теория X)» управления командой?

3) В чем преимущества и недостатки «Административной модели (теория X)» управления командой?
4) В чем состоит «модель хаоса (теория Y)» управления командой?
5) Укажите основные черты «модели хаоса (теория Y)» управления командой?
6) В чем преимущества и недостатки «модели хаоса (теория Y)» управления командой?
7) В чем состоит «открытая архитектура (теория Z)» управления командой?
8) Укажите основные черты «открытой архитектура (теория Z)» управления командой?
9) В чем преимущества и недостатки «открытой архитектуры (теория Z)» управления командой?

# Глава 12

# ОБЩЕНИЕ В КОМАНДЕ

Качество программ определяется продуктивностью обсуждения в группе, принимаемыми решениями и отклонениями от них. Основным фактором в разработке программного обеспечения является возможность коммуникации (общения участников проекта). Общение может проводиться в различных формах от строго формализованного (стандартизированная документация) до полностью неформализованного (вопрос-ответ соседу, обсуждение в неформальной обстановке).

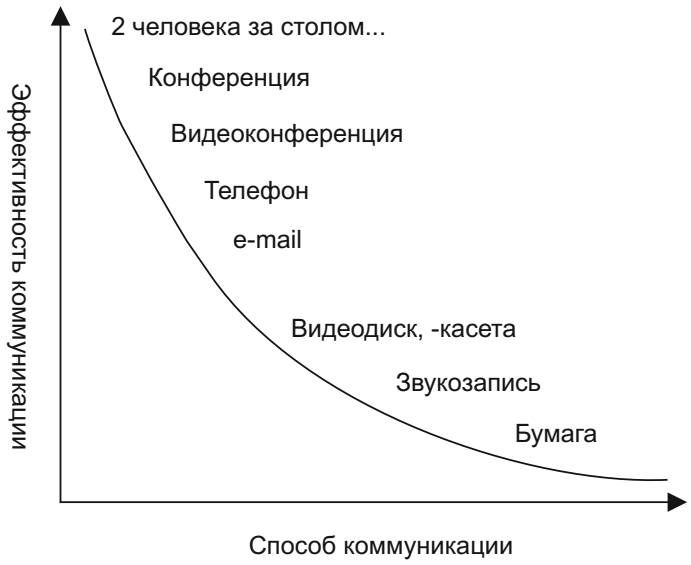



Рис. 12.1 – Эффективность различных видов общения для достижения заданной цели

На рисунке 12.1 изображена некая кривая, иллюстрирующая эффективность различных способов общения. Кривая является обобщением ряда исследований в этой области. Видно, что эффективность общения падает по мере возрастания степени его формализованности. С одной стороны, в этом нет ничего удивитель-

ного — старый принцип бюрократа гласит: хочешь получить отказ — пиши письмо, хочешь получить обещание — звони по телефону, хочешь добиться результата — езжай сам. Но с другой стороны, надо помнить, что коммуникации в команде определяются количеством участников (рабочих связей): при двух участниках — это одна связь, при n участниках — $n(n-2)/2$. При этом любая из этих связей может давать сбои и они не транзитивны: из того, что участник А хорошо контактирует с Б, а Б — с В, вовсе не следует, что А контактирует с В. То есть неформальное, «живое» общение эффективно только в относительно небольших, хорошо организованных (сработавшихся) коллективах.

Принятие решений в коллективе это — компромисс и консенсус. Целью общения в команде разработчиков являются обсуждение текущих проблем и вопросов и принятие решений. Далее мы рассмотрим некоторые проблемы организации обсуждений и принятия решений.

Итак, принятое в результате обсуждения решение может быть достигнуто в результате компромисса или в результате консенсуса. В чем разница этих результатов?

Начнем с определений (Глоссарий.ру):

*Компромисс* — соглашение, достигнутое посредством взаимных уступок.

*Консенсус* (коллективное мнение) — общее для конкретной группы мнение.

Компромисс: это среднее решение, которое может оказаться (и, как правило, оказывается) хуже каждого из вариантов. Достигается путем взаимных уступок (мы согласимся с вашим вариантом интерфейса, если вы согласитесь с нашей организацией базы данных). Может быть принят большинством (голосованием).

Консенсус: это оптимальное решение, сочетающее лучшее из предложенных вариантов. Достигается путем обсуждения, анализа и генерации новых идей. Принимается общим согласием (все согласны, что найдено лучшее решение).

Как добиться консенсуса? В отличие от компромисса, который чаще всего достигается в результате политических интриг и подковерных баталий, достижение консенсуса требует конструктивного и плодотворного напряжения всей команды и особого искусства управления командой. При этом рекомендуется придерживаться следующих принципов и правил:

Вера в достижение консенсуса — каждый член команды должен доверять другим в том, что обсуждение приведет к поиску оптимального решения, а не к борьбе личностных мнений. Создание такой атмосферы взаимного доверия является важнейшим в создании эффективной команды. Следует понимать, что взаимное доверие появляется не само по себе, а является результатом:

- Нескольких удачных консенсусов.
- Участием всех в выработке и принятии оптимальных решений.
- Созданием у каждого осознания причастности к принятым решениям.
- Выбора из вариантов решений, когда на обсуждении люди приходят не со сформированной позицией (ни шагу назад), а с вариантами возможных решений.
- Объективности принимаемых решений, как попытки ограничить проявления чувств и эмоций при обсуждении вопросов. Чувства и эмоции являются неотъемлемым свойством человеческой природы. Избежать их полно-

стью вряд ли удастся, но для приведения их «в норму» можно использовать следующие правила:

Критерий оценки вариантов — для объективности обсуждения крайне важно заранее договориться о критериях оценки — установить список критериев и выполнить их ранжировку по степени важности.

Разделение фактов и мнений. Факты — объективные показатели, выраженные в большинстве случаев количественно (но не обязательно): быстродействие, время отклика. Мнения — то, что не основано на фактах. Мнениями не следует пренебрегать, т. к. они часто основаны на опыте, интуиции.

Замена позиций — в случае, когда обсуждение все же заходит в тупик, бывает полезно предложить участникам изменить точку зрения: «перечислите, пожалуйста, сильные стороны варианта Вашего оппонента и слабые стороны Вашего варианта».

Легкости управления, когда роль руководителя в достижении консенсуса состоит в том, чтобы дать всем возможность высказаться и предложить свои варианты, оставляя свое мнение напоследок или не высказывать его совсем. Руководитель должен быть нейтрален. Руководитель (лидер) может принимать активное участие в обсуждении, но только на правах равного и поручить в этом случае руководство собранием другому человеку.

В силу свойственной IT-проектам неопределенности и рискованности одним из ключевых факторов их успеха являются эффективные компромиссные решения (trade-offs). Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (рамками). Эти три переменные образуют треугольник, показанный на рис. 12.2.


Рис. 12.2 – Треугольник компромисов

После достижения равновесия в этом треугольнике изменение на любой из его сторон для поддержания баланса требует модификаций на другой (двух других) стороне и/или на изначально измененной стороне. Нахождение верного баланса между ресурсами, временем разработки и возможностями — ключевой момент в построении решения, должным образом отвечающего нуждам заказчика. Изображенный выше треугольник помогает проиллюстрировать суть имеющихся ограничений и служит инструментом поиска компромиссных решений.

Реализуемая функциональность должна иметь качество не ниже определенного уровня. Параметр качества может быть изображен в виде четвертого измерения, превращающего треугольник в тетраэдр (треугольную пирамиду). Хотя снижение порога качества (quality bar) дает возможность сократить затрачиваемые ресурсы/время и увеличить функциональность решения, это ведет, безусловно, к неизбежно плохому результату.

Другое весьма полезное средство для управления проектными компромиссами — матрица компромиссов проекта (project tradeoff matrix), показанная в таблице 12.1. Она отражает достигнутое на ранних этапах проекта соглашение между проектной группой и заказчиком о выборе приоритетов в возможных в будущем компромиссных решениях. В определенных случаях из этой приоритезации могут делаться исключения, но в целом следование ей облегчает достижение соглашений по спорным вопросам.

Таблица 12.1 показывает матрицу компромиссов проекта, используемую обычно проектными группами Майкрософт. Она помогает обозначить проектное ограничение, воздействие на которое практически невозможно (колонка «Фиксируется»), фактор, являющийся в проекте приоритетным (колонка «Согласовывается»), и третий параметр, значение которого должно быть принято в соответствии с установленными значениями первых двух величин (колонка «Принимается»).

Нельзя произвольно сокращать функциональность решения. Как проектная группа, так и заказчик должны тщательно проанализировать все имеющиеся в проекте ограничения и быть готовыми идти на обусловленные ими уступки.

Таблица 12.1 – Матрица компромиссов

| **Треугольник компромисов** | Фиксируется | Согласовывается | Принимается |
|---|---|---|---|
| Ресурсы | **v** | | |
| Время | | **v** | |
| Возможности | | | **v** |

## Контрольные вопросы по главе 12

1) Как зависит качество ПО от общения в команде?
2) Приведите зависимость эффективности коммуникаций от способа коммуникации?
3) Укажите категории принятия решений в команде?
4) Дайте определение понятию «компромисс»?
5) Дайте определение понятию «консенсус»?
6) Приведите удачные варианты консенсусов?
7) В чем состоит «Треугольник компромисов»?

8) Что такое матрица компромиссов?
9) В чем состоит консенсунс «Объективность принимаемых решений» и «Критерии оценки вариантов»?
10) В чем состоит консенсунс «Разделение фактов и мнений» и «Замена позиций»?

# Глава 13

# КОРПОРАТИВНАЯ ПОЛИТИКА

***Корпоративная политика*** *— это внешние стратегии команд по учету влияния и воздействию на внешнюю среду вашего проекта. Это не только умение лидера проекта ладить с начальством, но и еще умение всей команды взаимодействовать друг с другом.*

Как правило, выделяются три измерения, в которых происходят эти взаимодействия: во властной структуре, в структуре задач и в информационной структуре.

- Взаимодействие во властной вертикали — это создание репутации, получение поддержки со стороны руководства, получение лучших проектов, оборудования и софта, «прикрытие» от политических бурь. Человек, выполняющий все это, должен быть политиком, ориентирующимся в кабинетах власти фирмы.
- Координация задач выполняется в горизонтальной плоскости и состоит во взаимодействии с другими подразделениями фирмы. Координаторы обеспечивают поступление проекта и его сдачу, взаимодействие с внешними тестерами, изучение интерфейса с группой анализа человеческого фактора, получение и передачу библиотек компонентов, договариваются с другими группами, выторговывая ресурсы и услуги.
- Взаимодействие в информационной структуре предполагает исследование и сбор информации, необходимой для успешного выполнения проекта. Информационные исследователи ищут нужное и просеивают поступающее.

В разных командах складываются разные стили игры в корпоративную политику. Чему следует отдавать предпочтение? Проводились исследования (Дебора Анкона — Deborah Ancona) эффективности команд четырех типов: политики, изоляционисты, исследователи и универсалы. В начале исследования политики и изо-

ляционисты считали себя лучше других, хотя с точки зрения руководства лучшими выглядели политики и универсалы. Через полгода оказалось, что самые низкие оценки производительности — у политиков и исследователей (первые много говорили, вторые много собирали информации, но и те и другие мало что делали). Далее шли изоляционисты. Здесь результаты были неоднозначны. Часть команд пришла к полному провалу, часть получила ощутимые результаты. Лучшие результаты были у универсалов.

## Контрольные вопросы по главе 13

1) Что такое «Корпоративная политика»?
2) Сколько выделяется измерений, в которых происходят взаимодействия в команде?
3) Что такое «Взаимодействие во властной вертикали»?
4) Что такое «Координация задач»?
5) Что такое «Взаимодействие в информационной структуре»?
6) Укажите стили корпоративной политики?

# Глава 14

# ПЛАНИРОВАНИЕ И КОНТРОЛЬ

Проект должен быть предсказуемым как по качеству выполняемых работ, так и по своевременности выполнения отдельных этапов (жизненный цикл программы). План — один из основных элементов предсказуемости проекта. Контроль хода выполнения плана позволяет оценить возможность продолжения проекта. Проект имеет элемент неопределенности. То есть заранее всего предусмотреть нельзя, в силу чего первоначальные планы обычно и не выполняются. Но при возникновении ранее неучтенных обстоятельств планы можно и нужно корректировать для сохранения предсказуемости проекта.

Все возможные особенности: технические, производственные, человеческие необходимо учитывать при планировании. Основными функциями планирования являются:

- Преобразование потребностей в управляемые задачи.
- Изначально проект выступает в виде требований, разработанных и согласованных с Заказчиком. Цель планирования — представить его в виде совокупности отдельных задач, выполнение которых можно контролировать.
- Определение необходимых ресурсов.
- Детальные планы позволят Вам определить количество людей, необходимого оборудования и рабочие условия, которые понадобятся для выполнения проекта.
- Координация командной работы над проектом.
- Очень часто выполнение проекта разбивается на отдельные работы, которые можно выполнять параллельно. Планы делают возможной координацию путем определения того, кто, что и когда делает.
- Оценка потенциальных рисков.
- Хотя некоторые риски могут быть выявлены во время формулировки требований, гораздо больше их обнаруживается после осуществления деталь-

ного планирования. Знание о существовании этих рисков позволит Вам раньше их заметить (если они осуществились) и приготовиться к их адресации.

- Сигнализация о возникновении проблем.
- Отклонение от плана — сигнал о возникновении проблемы. Планы — это не догма, которой необходимо безоговорочно следовать. Для менеджера проекта они скорее являются предположениями и основой для сравнения. Если выполнение проекта не оправдывает ожиданий, то необходимо провести соответствующую корректировку плана.

## Контрольные вопросы по главе 14

1) Что такое план выполнения проекта?
2) Какие неопределнности имеет проект?
3) Укажите основные функции планирования проекта?

# Глава 15

# ДЕКОМПОЗИЦИЯ ВИДОВ РАБОТ

Важнейшим элементом планирования является разбиение проекта на отдельные задачи, подзадачи и действия с дальнейшей оценкой сроков, ресурсов и порядка их выполнения. Этот элемент планирования называют структурной декомпозицией работ (СДР, или WBS — Work Breakdown Structure).

***СДР*** *— это иерархическая декомпозиция и организация деятельностей (задач, подзадач, действий), необходимых для удовлетворения целей проекта.*

Организация и уровень детализации деятельности будут способствовать оценке, распределению работ и дальнейшему управлению. СДР помогает сделать цели проекта управляемыми. Хотя проект может состоять всего из нескольких сотен задач, но уже ими практически невозможно будет руководить, если они будут находиться в одной куче. СДР служит идее организации задач с целью упрощения работ по оцениванию, распределению, координированию и пересмотру.

На деятельностях, определенных в СДР, базируются планы проекта, включая:

1) Календарный план-график проекта.
2) План распределения ресурсов.
3) Бюджетный план.
4) План управления качеством.
5) План управления рисками.

Ниже перечислены основные шаги процесса, которому можно следовать при построении СДР:

- Определить основные цели проекта.

- Определить функциональные требования, которые удовлетворяют целям проекта.
- Определить основные задачи, соответствующие функциональным требованиям. Чтобы достигнуть более высокой управляемости проекта и убедиться в том, что вы включили все существенные задачи, часто полезно включить промежуточный уровень классификации.

Можно систематизировать задачи, используя предлагаемые уровни группировки или их комбинации:

- системы (основные аппаратные и программные подсистемы);
- этапы или фазы (как концепция, инициация, проектирование, разработка, сборка и тестирование);
- организации (отделы и географические дислокации).

Подразделяйте основные задачи на более мелкие, которые будут отражать то, каким образом планируется завершить работу.

Составьте графическую схему, создавая столько слоев, сколько необходимо для полного разбиения объема работ на достаточно малые управляемые части с уровнем детализации, который позволяет:

- оценивать работы и определять их временные рамки;
- назначать работы исполнителям (группам);
- видеть и обсуждать продвижение работ.

Для достижения поставленных целей (оценка, распределение и контроль выполнения работ) СДР должна удовлетворять следующим критериям:

- целенаправленности. Все деятельности должны быть направлены на достижение единой цели проекта и вести к конечному результату;
- независимости. Между деятельностями в рамках проекта очень часто существует множество зависимостей. Управлять и контролировать такие деятельности достаточно сложно. Для повышения управляемости проектом любая деятельность должна быть определена до такого уровня детальности, что она будет завершена без необходимости активной координации с результатами других деятельностей. Такой ситуацией лучше всего управлять путем разбиения работы на равнозависимые подмножества;
- определенности продолжительности. Деятельности не должны быть «безлимитными» во времени, так как в этом случае они непременно растянутся, выходя за пределы самых худших ожиданий. Длительность также косвенно устанавливает ожидаемое качество результата;
- четкости понимания. Деятельность должна предполагать результат, однозначно понимаемый людьми, которые будут выполнять эти работы. Результатом может быть замысел, решение, документ, тест и т. д. Результат должен быть представлен в четко понимаемой форме (общее описание, документ по шаблону, исходный код в соответствии с принятыми правилами оформления, . . .);

- достижимости. Планируемый результат должен быть достижим: установленные сроки, выделяемые ресурсы, квалификация исполнителей, организация работ должны быть реальны и достаточны для планируемых результатов отдельных деятельностей с учетом предполагаемого уровня качества;
- отработанности. Большинство разрабатываемых проектов сопровождаются созданием чего-то нового. Тем не менее работы, которые приводят к этому созданию, в основном уже проводились ранее (возможно, немного по другому, чем это понадобится для нового проекта). Отработанность детальных задач способствует оценке и распределению работ.

На рис. 15.1 приведена декомпозиция видов работ при разработке программного обеспечения. Из рисунка видно, что первым определяющим шагом при разработке является выбор цели. Далее формулируются функции разработки отдельно для будущей программы, аппаратуры (операционная система, звуковая и видеоплаты, вывод на экран и принтер и др.) и др. Проработка подфункции, например программное обеспечение (ПО), делится на задачи (выбор операционнай системы, языка программирования, технологии программрования, соответствующих баз данных и др.).

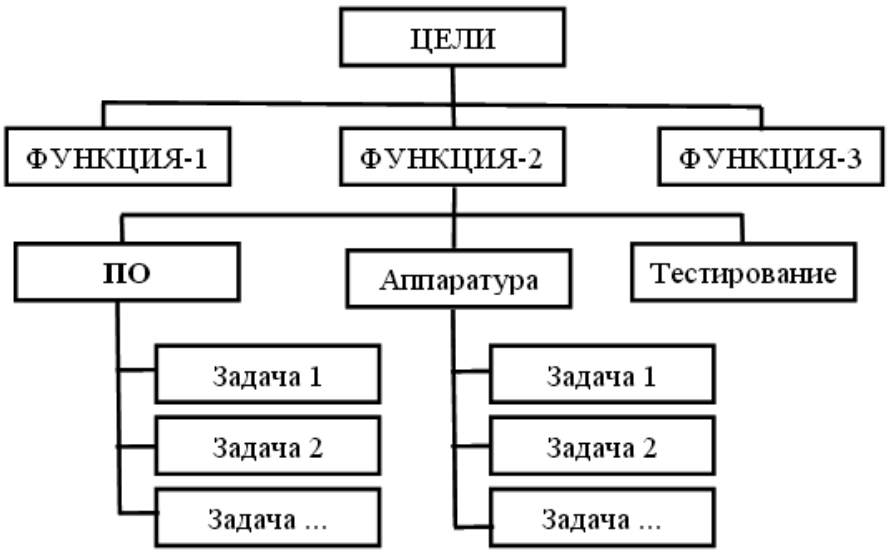



Рис. 15.1 – Декомпозиция видов работ

В настоящее время вышли несколько международных стандартов по планированию проектов:

IEEE Std 1058–1998 «IEEE Standard for Software Project Management Plans».

Plan Content Содержание плана.

IEEE Std. 1228–1994. IEEE Standard for Software Safety Plans.

IEEE Std. 1059–1993. IEEE Guide for Software Verification and Validation Plans.

IEEE Std. 730–2002. IEEE Standard for Software Quality Assurance Plans.

IEEE Std. 828–1998. IEEE Standard for Software Configuration Management Plans.

## Контрольные вопросы по главе 15

1) Что такое структурная декомпозиция работ и какие планы существуют при структурной декомпозиции работ?
2) Укажите основные шаги, которые необходимо выполнить при структурной декомпозиции работ?
3) Какие возникают задачи при структурной декомпозиции работ?
4) Каким критериям должна удовлетворять структурная декомпозиции работ?
5) В чем состоит критерий структурной декомпозиции работ «Целенаправленность»?
6) В чем состоит критерий структурной декомпозиции работ «Независимость»?
7) В чем состоит критерий структурной декомпозиции работ «Определенность продолжительности»?
8) В чем состоит критерий структурной декомпозиции работ «Четкость понимания» и «Достижимость»?
9) В чем состоит критерий структурной декомпозиции работ «Отработанность»?

# Глава 16

# СРЕДСТВА УПРАВЛЕНИЯ ПРОЕКТОМ

Система календарного планирования позволяет руководителю компании понять, как эффективно используются ресурсы, а также помогает при планировании видеть ясную картину происходящего.

Компьютерная система управления проектом обеспечивает доступ к информации, минуя бюрократические и географические барьеры. Участники проекта смогут иметь доступ к проектной информации в режиме реального времени, даже если они находятся далеко друг от друга. Система позволяет получить общее представление обо всех проектах, которые имеются в портфеле проектов, наглядно отображает взаимосвязи между задачами, позволяет вовремя заметить проблемы. Средства визуального отображения позволяют организовать обмен информацией между членами проектной команды и клиентами.

Рынок программных продуктов для управления проектами растет и развивается. Системы все в большей степени ориентированы на Интернет. Такие системы позволяют обеспечить доступ к проектной документации для всех членов команды в режиме реального времени. В будущем все больше организаций будут использовать программные продукты. Ожидается, что в будущем системы от различных производителей будут все более похожи друг на друга. Это объяснятся тем, что базовая основа всех систем календарного планирования одна и та же. Кроме того, пользователи хотят видеть проектные данные в привычном и понятном им формате.

Инструментальные средства управления проектом должны поддерживать следующие основные функции:

1. Комплекс работ, связей и временных характеристик. Средства описания комплекса работ проекта, связей между работами и их временных характеристик должны включать:

- описания глобальных параметров планирования проекта;

- описание логической структуры комплекса работ;
- многоуровневое представление проекта;
- назначение временных параметров планирования задач;
- поддержку календарей отдельных задач и проекта в целом.

2. Информация о ресурсах и затратах. Средства поддержки информации о ресурсах и затратах по проекту и назначения ресурсов и затрат отдельным работам проекта должны обеспечивать решение следующих задач:

- организационная структура исполнителей;
- ведение списка наличных ресурсов, номенклатуры материалов и статей затрат;
- поддержку календарей ресурсов;
- назначение ресурсов работам;
- календарное планирование при ограниченных ресурсах.

3. Контроль за ходом выполнения. Средства контроля за ходом выполнения проекта должны обеспечивать:

- фиксацию плановых параметров расписания проекта в базе данных;
- ввод фактических показателей состояния задач;
- ввод фактических объемов работ и использования ресурсов;
- сравнение плановых и фактических показателей и прогнозирование хода предстоящих работ.

4. Представление структуры проекта, отчетов. Графические средства представления структуры проекта, средства создания различных отчетов по проекту в виде:

- диаграммы Гантта (часто совмещенной с электронной таблицей и позволяющей отображать различную дополнительную информацию);
- PERT-диаграммы (сетевая диаграмма);
- создания отчетов, необходимых для планирования и контроля.

5. Дополнительные программные продукты. «Классические» системы календарного планирования в последнее время дополняются программными продуктами, которые позволяют:

- добавить или улучшить отдельные функции управления проектами, например анализ рисков, учет рабочего времени исполнителей, расчет расписания при ограниченных ресурсах;
- интегрировать системы управления проектами в корпоративные управленческие системы;
- настроить универсальное программное обеспечение на специфику управления проектами в конкретной предметной области (например, интеграция со сметными системами для строительных проектов).

Приведем краткий обзор систем управления проектами. К числу наиболее известных систем для управления проектами относятся:

1. *MS Excel.* Хорошо подходит для недельного планирования и отчетности.

2. *MS Project.* Microsoft Project является на сегодня самой распространенной в мире системой управления проектами. Во многих западных компаниях MS Project стал привычной добавкой к Microsoft Office даже для рядовых сотрудников, которые используют его для планирования графиков несложных комплексов работ. Отличительной особенностью пакета является его простота. Разработчики MS Project не стремятся вложить в пакет сложные алгоритмы календарного или ресурсного планирования.

Семейство программ состоит из следующих продуктов:

*MS Project Standard* — рус. Легкая, универсальная система для управления проектами, в том числе для планирования и формирования графиков выполнения проектов. В сочетании с сервером MS Project Server позволяет наладить коллективную работу над проектом в масштабах рабочей группы на предприятиях разной величины.

*MS Project Professional.* Новое приложение. Содержит всю функциональность Microsoft Project Standard и в сочетании с Microsoft Project Server обеспечивает поддержку коллективной работы над проектами и предоставляет средства анализа и управления проектами и ресурсами в масштабах крупного предприятия.

*MS Project Server* — рус. Очередное пополнение в семействе Microsoft .NET Server, которое в сочетании с Microsoft Project Professional и Microsoft Project Standard обеспечивает полноценную поддержку коллективной работы над проектами, а также содержит средства анализа и управления ресурсами в масштабах всего предприятия.

*MS Project Web Access.* Web-интерфейс, предоставляющий доступ к информации о проектах и средствам анализа для руководителей и членов групп, которым не нужен полный набор функций управления в Microsoft Project. Эти пользователи могут обращаться к информации о проектах через Web-браузер. Поставляется в составе Microsoft Project Server.

3. *Open Plan.* Производитель Welcom Corp. (США). Дистрибьютор в России ЛАНИТ. Open Plan — полностью руссифицированная система планирования и контроля крупных проектов и программ. Основные отличия системы от аналогов:

- мощные средства ресурсного и стоимостного планирования,
- эффективная организация многопользовательской работы,
- возможность создания открытого, масштабируемого решения для всего предприятия.

Open Plan поставляется в двух вариантах — Professional и Desktop — каждый из которых отвечает различным потребностям исполнителей, менеджеров и других участников проекта.

*Primavera Project Planner.* Центральный программный продукт семейства Primavera, Primavera Project Planner (P3) применяется для календарно-сетевого планирования и управления с учетом потребностей в материальных, трудовых и финансовых ресурсах средними и крупными проектами в самых различных областях, хотя наибольшее распространение данный продукт получил в сфере управления строительными и инженерными проектами.

*SureTrak Project Manager.* Кроме P3, компанией Primavera Systems поставляется облегченная система для УП — SureTrak. Этот полностью русифицированный

продукт ориентирован на контроль выполнения небольших проектов или/и фрагментов крупных проектов. Может работать как самостоятельно, так и совместно с P3 в корпоративной системе управления проектами.

4. *Spider Project.* Производитель Spider Technologies Group (Россия). Российская разработка Spider Project отличается мощными алгоритмами планирования использования ограниченных ресурсов и большим количеством дополнительных функций. Система спроектирована с учетом большого практического опыта, потребностей, особенностей и приоритетов Российского рынка. Spider Project поставляется в двух вариантах – Professional и Desktop.

5. *Project Expert.* Производитель Про-Инвест Консалтинг (Россия). Российская разработка Project Expert обеспечивает построение финансовой модели предприятия, анализ финансовой эффективности бизнес-проектов, разработку стратегического плана развития и подготовку бизнес-плана.

6. *1С-Парус: Управление проектами.* 1С-Парус (Россия). Российская разработка на платформе бухгалтерской системы «1С:Предприятие » версии 7.7 служит для планирования, организации, координации и контроля проектных работ и ресурсов. Типовое решение разработано только средствами и методами программы «1С: Предприятие » и представляет собой дополнение к компоненте «Бухгалтерский учет» программы «1С:Предприятие» версии 7.7. 1С-Парус:Управление проектами интегрируется с любыми конфигурациями, которые используют компоненту 1С «Бухгалтерский учет».

7. *Капитал CSE* (система управления ресурсами предприятия) российского предприятия Геликон-Про (http://www.gelicon.biz). Программный продукт позволяет выполнять следуюущие функции:

- управление производством,
- управление финансами,
- бухгалтерский и налоговый учет,
- учет основных фондов,
- учет движения материальных ценностей,
- управление персоналом,
- расчет заработной платы,
- управление снабжением и сбытом.

## Контрольные вопросы по главе 16

1) Что такое Система календарного планирования?
2) Укажите основные функции, которые необходимо поддерживать при управлении проектом?
3) В чем заключается функция управления проектом «Комплекс работ, связей и временных характеристик»?

4) В чем заключается функция управления проектом «Информация о ресурсах и затратах» и «Контроль за ходом выполнения»?
5) В чем заключаются функции управления проектом и «Представление структуры проекта, отчетов» и «Дополнительные программные продукты»?
6) Для каких целей применяется MS Excel?
7) Для каких целей применяется MS Project, MS Project Standard?
8) Для каких целей применяется Open Plan и Primavera Project Planner?
9) Для каких целей применяется SureTrak Project Manager и Spider Project?
10) Для каких целей применяется Project Expert и 1С-Парус: Управление проектами?

# Глава 17

# УПРАВЛЕНИЕ КАЧЕСТВОМ ПРОЕКТА

Любое изделие, в том числе программный продукт, должно быть качественным. В различных источниках можно найти различные определения качества [13]:

- Качество — все, что составляет сущность лица или вещи (Словарь Даля).
- Качество — философская категория, выражающая неотделимую от бытия объекта его существенную определенность, благодаря которой он является именно этим, а не иным объектом (Большая Советская Энциклопедия).
- Качество продукции — совокупность свойств продукции, обусловливающих ее способность удовлетворять определенные потребности в соответствии с ее назначением (БСЭ).
- Качество товара — совокупность потребительских свойств товара (ГОСТ Р 51303–99).

*Качество* — это свойство товара (услуги) наиболее полно удовлетворять требованиям и пожеланиям потребителя.

Наиболее общим является подход определения качества, при котором вводятся понятия:

- Ценность изделия — способность удовлетворять потребности.
- Качество изделия — соответствие между свойствами изделия и его ценностью.
- Мера качества — соотношение ценности и стоимости.

При этом оказывается (см. рис. 17.1), что для производителя и потребителя эти показатели имеют различные значения. Для производителя вся продукция, не содержащая дефектов, которые препятствовали бы продаже этой продукции, имеет ценность. Для потребителя же ценность имеют только те свойства продукции, которые соответствуют его ожиданиям. Важными являются три основных соотношения между ценностью и стоимостью:

Мера качества для потребителя: $Q_u = C_u / S_u$.

Мера качества для производителя: $Q_d = C_d / S_d$.

Конкурентоспособность продукта: $K = C_u / C_d$.

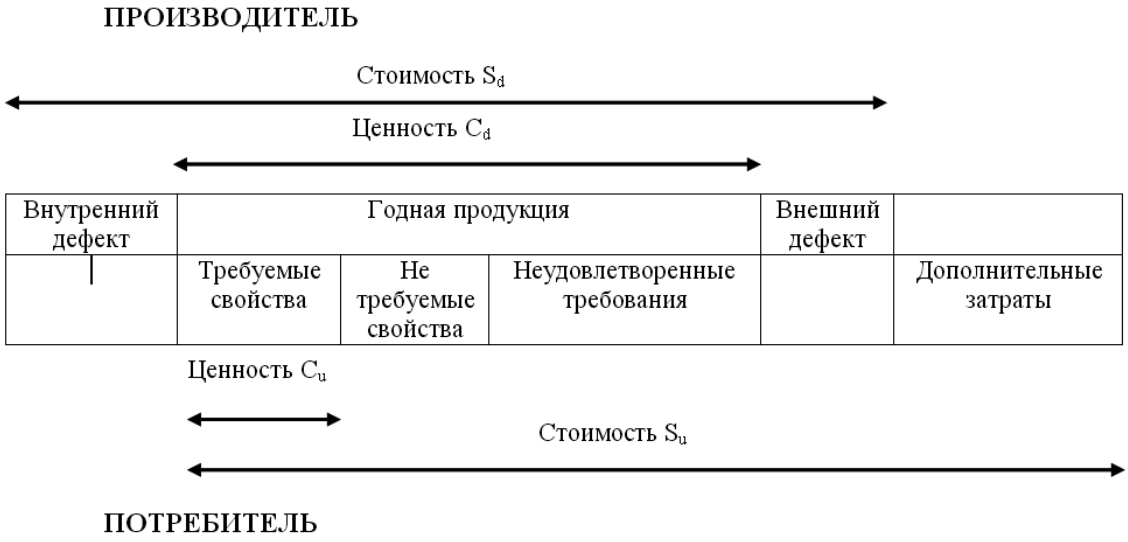



Рис. 17.1 – Соотношение цены и стоимости со стороны Производителя продукции и Потребителя

На разных этапах развития производства, в том числе и программных продуктов, применяются различные методы обеспечения качества. Выделяют три основные фазы эволюции методов обеспечения качества:

- фаза отбраковки;
- фаза управления качеством;
- фаза планирования качества.

*Фаза отбраковки* — Началась вместе с зарождением ремесленного производства. Отдельные мастера проверяли свою собственную работу. Цеховые организации средневековых городов, которые, если выражаться современным языком, сертифицировали мастеров, — присуждали звание мастера после серьезных испытаний качества изделия. При этом каждое изделие было индивидуальным. Выходной контроль вместо входного начали делать в начале XX века. Форд впервые ввел вместо входного контроля комплектующих на сборке выходной контроль на тех производствах, где эти комплектующие изготавливались, то есть на сборку стали поступать только годные, качественные изделия. Он также создал отдельную службу технического контроля, независимую от производства.

Концепция фазы отбраковки состоит в том, что потребитель должен получать только годные изделия, т. е. изделия, соответствующие стандартам. Основные усилия должны быть направлены на то, чтобы негодные изделия (брак) были бы отсечены от потребителя.

*Фаза управления качеством* — Цель фазы управления качеством — сосредоточить усилия не на том, как обнаружить и изъять негодные изделия до их отгрузки покупателю, а на том, как увеличить выход годных изделий в техпроцессе. В этой фазе выделяют два этапа:

- Управление процессами — переход от контроля к управлению отдельными процессами.

- Управление производством — переход от управления отдельными процессами к управлению производством в целом.

Точкой отсчета этого направления считаются работы, выполненные в Отделе технического контроля фирмы Вестерн Электрик, США. В мае 1924 г. сотрудник отдела доктор Шухарт передал своему начальнику короткую записку, которая содержала метод построения диаграмм, известных нынче по всему миру как контрольные карты Шухарта. Контрольные карты основаны на статистических методах оценки стабильности протекания различных технологических процессов. На основе выборок — замеров контрольных показателей процессов (к-ва брака в контрольной партии) — строится среднее значение и допустимые верхнее и нижнее отклонения. Процесс не должен выходить за допустимые значения. Статистические методы, предложенные Шухартом, дали в руки управленцев инструмент, который позволил контролировать качество производства комплектующих.

Для первого этапа было характерно создание аудиторских служб по качеству, которые, в отличие от отделов технического контроля, занимались не разбраковкой продукции, а путем контроля небольших выборок из партий изделий проверяли работоспособность системы обеспечения качества на производстве.

Внедрение таких служб контроля качества отдельных процессов значительно повысило эффективность производства, но есть предел, определяемый системой. Каждый производственный процесс имеет определенный предел выхода годных изделий, и этот предел определяется не процессом самим по себе, а системой, то есть всей совокупностью деятельности предприятия, организации труда, управления, в которой этот процесс протекает.

*Второй этап управления качеством* (фаза менеджмента качества) связан с повышением качества путем управления предприятием. Составными элементами этого этапа являются:

- Совершенствование системы в целом, а не только отдельных производственных процессов.
- Непосредственное участие высшего руководства компаний в проблемах качества.
- Обучение всех сотрудников компаний сверху донизу основным методам обеспечения качества.
- Упор на мотивацию сотрудников на высококачественный труд.

*Фаза планирования качества*. Цель — планирование запросов. Эта фаза стала зарождаться в середине 60-х гг. как развитие идей в направлении более полного удовлетворения запросов потребителей.

Большая часть дефектов изделий закладывается на стадии разработки из-за недостаточного качества проектных работ. Поэтому происходит тенденция переноса центра тяжести работ по созданию изделия с натурных испытаний опытных образцов или партий на математическое моделирование свойств изделий, а также моделирование процессов производства изделий, что позволяет обнаружить и устранить конструкторские и технологическое дефекты еще до начала стадии производства. Задача моделирования состоит в снижении цены изделия при сохранении высокого качества и конкурентноспособности.

Согласие, достигнутое по требованиями к качеству (quality requirements), наравне с четким доведением до программных инженеров того, что составляет качество получаемого продукта, требует обсуждения и формального определения многих аспектов качества. Важной идеей является то, что программные требования определяют требуемые характеристики качества программного обеспечения, а также влияют на методы количественной оценки и сформулированные для оценки этих характеристик соответствующие критерии приемки.

*1. Культура и этика программной инженерии (Software Engineering Culture and Ethics).* Ожидается, что инженеры по программному обеспечению воспринимают вопросы качества программного обеспечения как часть своей профессиональной культуры. Этические аспекты могут играть значительную роль в обеспечении качества программного обеспечения, культуре и отношении инженеров к своей работе.

*2. Значение и стоимость качества (Value and Costs of Quality).* Понятие «качество», на самом деле, не столь очевидно и просто, как это может показаться на первый взгляд. Для любого инженерного продукта существует множество интерпретаций качества, в зависимости от конкретной «системы координат». Множество этих точек зрения необходимо обсудить и определить на этапе выработки требований к программному продукту. Характеристики качества могут требоваться в той или иной степени, могут отсутствовать или могут задавать определенные требования, все это может быть результатом определенного компромисса. Стоимость качества (cost of quality) может быть дифференцирована на стоимость предупреждения дефектов (prevention cost), стоимость оценки (appraisal cost), стоимость внутренних (internal failure cost), а также внешних сбоев (external failure cost).

Движущей силой программных проектов является желание создать программное обеспечение, обладающее определенной ценностью. Ценность программного обеспечения может выражаться в форме стоимости, а может и нет. Заказчик, обычно, имеет свое представление о максимальных стоимостных вложениях, возврат которых ожидается в случае достижения основных целей создания программного обеспечения. Заказчик может также иметь определенные ожидания в отношении качества ПО. Иногда заказчики не задумываются о вопросах качества и связанной с ними стоимостью.

*3. Модели и характеристики качества (Models and Quality Characteristics).* В различных источниках (таксономиях и моделях) терминология характеристик качества программного обеспечения отличается. Каждая модель включает различное число уровней иерархии и общее число «распознанных» характеристик качества. Различные авторы создали разные модели качества со своим набором характеристик и атрибутов. Эти модели могут быть полезны для обсуждения, планирования, адаптации и оценки качества программных продуктов.

*4. Повышение качества (Quality Improvement).* Качество программного обеспечения может повышаться за счет итеративного процесса постоянного улучшения. Это требует контроля, координации и обратной связи в процессе управления многими одновременно выполняемыми процессами: (1) процессами жизненного цикла, (2) процессом обнаружения, устранения и предотвращения сбоев/дефектов и (3) процессами улучшения качества. К программной инженерии применимы теории и концепции, лежащие в основе совершенствования качества. Например, предотвращение и ранняя диагностика ошибок, постоянное совершенствование

(continuous improvement) и внимание к требованиям заказчика (customer focus), составляющие принцип «building in quality». Эти концепции основываются на работах экспертов по качеству, пришедших к мнению, что качество продукта напрямую связано с качеством используемых для его создания процессов.

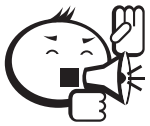
## Контрольные вопросы по главе 17

1) Дайте определение понятию «качество»?
2) Приведите выражение для расчета «Конкурентоспособность продукта»?
3) Приведите выражения для расчета «Мера качества для потребителя» и «Мера качества для производителя»?
4) Приведите основные фазы эволюции методов обеспечения качества?
5) В чем состоят фазы обеспечения качества «Фаза отбраковки», «Фаза управления качеством» и «Фаза планирования качества»?
6) Приведите этапы управления качеством?
7) Раскройте сущность фазы управления качеством «Фаза планирования качества»?
8) Приведите требуемые характеристики качества программного обеспечения?
9) В чем состоит сущность характеристики качества «Культура и этика программной инженерии» и «Значение и стоимость качества»?
10) В чем состоит сущность характеристики качества «Модели и характеристики качества» и «Повышение качества»?

# Глава 18

# КАЧЕСТВО ПО И МЕТОДЫ ЕГО ОБЕСПЕЧЕНИЯ

Требования к ПО определяют, какие свойства и характеристики оно должно иметь для удовлетворения потребностей пользователей и других заинтересованных лиц.



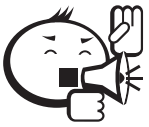
Для того, чтобы ПО действительно было полезным, важно, чтобы отражались их реальные потребности, которые часто отличаются от непосредственно выражемых пользователями и заказчиками желаний.

Для выявления этих потребностей зачастую приходится проводить достаточно значительную работу. При разработке ПО для непосредственной продажи это практически очевидно — чтобы оно действительно продавалось, необходимо сначала выяснить, что же хотят его потенциальные покупатели. Но и при разработке на заказ, когда заказчик приходит с вроде бы готовым заданием — например, ему нужно сделать систему поддержки контроля качества строительства и эксплуатации магистральных газопроводов — полезно выяснить, в чем же именно он нуждается, какие основные задачи должна решать эта система, какие из задач стоят очень остро и обязательно должны быть отражены уже в первой версии, а какие могут и потерпеть.

Определение потребностей базируется на анализе предметной области, выявляющего основные цели и задачи деятельности организаций-будущих пользователей системы. Потребности выявляются на основе наиболее актуальных проблем и задач, которые эти организации видят перед собой. При этом часто требуется аккуратное рассмотрение текущей ситуации, выявление значимых проблем и расстановка приоритетов их решения, поскольку чаще всего решить сразу все проблемы невозможно.

Формулировка потребностей может быть разбита на следующие этапы:

1) Выделить одну-две-три основные проблемы.
2) Определить причины возникновения проблем, оценить их степень влияния и выделить наиболее существенные из проблем, влекущие появление остальных.
3) Выявить всех заинтересованных лиц, которым придется иметь дело с системой.
4) Определить границы системы — линию раздела между системой, которую надо сделать, и ее окружением, влияние которого надо учитывать.
5) Определить ограничения на возможные решения.



Формулировка потребностей не должна накладывать лишних ограничений на возможные решения, удовлетворяющие им.

Нужно попытаться сформулировать, что именно является проблемой, а не придумывать сразу возможные решения.

Например, формулировки «система должна использовать СУБД Oracle для хранения данных», «нужно, чтобы при вводе неверных данных раздавался звуковой сигнал» не очень хорошо описывают потребности (за исключением особых случаев, например если использование конкретного поставщика СУБД является внешним ограничением, политикой компании, при этом первая формулировка должна быть взята за основу). Соответствующие потребности лучше описать так: «нужно организовать надежное хранение данных», «необходимо предотвращать попадание некорректных данных в хранилище».

При выявлении потребностей пользователей используются такие приемы, как анкетирование, демонстрация раскадровок работы будущей системы, интерактивные сеансы, где пользователям предоставляется возможность самим предложить варианты внешнего вида системы и ее работы или поменять предложенные кем-то другим прототипы.

На основе потребностей пользователей формулируются возможные функции будущей системы, представляющие собой услуги, удовлетворяющие потребности одной или нескольких групп пользователей (или других заинтересованных лиц). Идеи для определения функций можно брать из имеющегося опыта (наиболее часто используемый источник) или из результатов мозговых штурмов и других форм выработки.

Формулировка функций должна быть достаточно короткой, ясной для пользователей, без лишних деталей. Например:

- Все данные измерений будут сохраняться в базе данных.
- Статус выполнения заказа можно будет узнать через Интернет.
- Система будет поддерживать до 1000 одновременно работающих пользователей.
- Расписание проведения ремонтных работ будет строится автоматически.

Только имея набор функций, достаточно полно решающий проблемы, выделенные на предыдущем этапе как наиболее существенные, можно составлять требования — детализацию работы этих функций. При этом надо учитывать, что часто ПО является частью программно-аппаратной системы, требования к которой надо преобразовать на требования к программной и аппаратной ее составляющим (в последнее время, в связи со значительным падением цен на мощное аппаратное обеспечение общего назначения фокус внимания переместился, в основном, на программное обеспечение). Соотношение между проблемами, потребностями, функциями и требованиями изображено на рис. 18.1.


Рис. 18.1 – Соотношение между проблемами, потребностями, функциями и требованиями

Каждое требование раскрывает детали поведения системы при выполнении ею некоторой функции в некоторых обстоятельствах. При этом часть требований происходит из пожеланий заинтересованных лиц и решений, удовлетворяющих эти пожелания, а часть — из внешних ограничений, накладываемых на систему, например, законами той области знания, в рамках которой системе придется работать, государственным законодательством, корпоративной политикой и пр.

Еще до перехода от функций к требованиям полезно расставить приоритеты и оценить трудоемкость их реализации и рискованность. Это позволит отказаться от реализации наименее важных функций еще до их детальной проработки, а также выявить наиболее проблемные места проекта.

Список стандартов, определяющих правила работы с требованиями к ПО (и более общими, системными требованиями), выглядит так. IEEE 830–1998 Recommended Practice for Software Requirements Specifications. Описывает структуру документов для фиксации требований к ПО. Кроме того, он определяет характеристики, которыми должен обладать правильно составленный набор требований.

- Корректность (соответствие реальным потребностям).
- Недвусмысленность (однозначность понимания).
- Полнота (отражение всех основных потребностей).
- Непротиворечивость (согласованность между различными элементами).
- Упорядоченность по важности и стабильности.
- Возможность проверки.

- Модифицируемость.
- Прослеживаемость в ходе разработки (возможность увязать требование с подсистемами, модулями и операциями, ответственными за его выполнение, и с тестами, проверяющими его выполнение).

## Контрольные вопросы по главе 18

1) Что является важным, чтобы ПО действительно было полезным?
2) Что такое потребности?
3) Укажите основные этапы формулировки потребностей?
4) Что формулирует пользователь на основе потребностей?
5) На каком этапе выделяются требования к ПО?
6) Приведите графически схему соотношения между проблемами, потребностями, функциями и требованиями?
7) Какие требования выдвигаются при разработке ПО согласно стандарту IEEE 830-1998?

# Глава 19

# РИСКИ

Дисциплина управления рисками MSF возводит процесс управления рисками в ранг стратегической задачи, затрагивающей все фазы проекта. В рамках MSF управление рисками — это процесс выявления, анализа и превентивной работы над рисками в целях избежания их превращения в проблемы, приносящее ущерб или иной вред. В ходе всего проекта команда должна уделять внимание дисциплине управления рисками.

***Риск проекта*** *(project risk) понимается в MSF — как всякое событие или условие, которое может оказать как негативное, так и позитивное влияние на итоги проекта.*

Важно отметить, что риски не есть проблемы. Проблемы — это нечто, имеющее место в настоящее время, в то время как риски относятся к будущему и носят вероятностный характер (могут и не состояться). Однако риски могут стать проблемами, если ими эффективно не управлять.

Цель управления рисками — максимизировать их положительное влияние (открывающиеся возможности), но при этом минимизировать связанные с ними негативные факторы (убытки). Дисциплина управления рисками в MSF основана на убеждении, что такое управление должно выполняться превентивно; это часть формального и систематического процесса, трактующего усилия, затрачиваемые на управление рисками, с позитивной точки зрения.

Говоря о рисках, MSF выделяет несколько ключевых концепций:

1) *Риск* — неотъемлемая часть всякого проекта или процесса. Несмотря на то, что различные проекты могут быть связаны с большим или меньшим числом рисков, не существует ни одного проекта, полностью свободного от них. Цель состоит не в том, чтобы избежать рисков, а в том, чтобы пред-

восхищать потенциальные проблемы и заблаговременно готовиться к их решению, если они возникнут.

2) *Выявление рисков нужно всячески одобрять*. Проектная группа должна смотреть на выявление рисков как на позитивную деятельность. Знание о существовании рисков — необходимое условие эффективной работы над ними. Следовательно, перспективы успеха проектной группы от проведения работы по выявлению рисков лишь увеличиваются.

3) *Оценка рисков должна вестись постоянно*. Обстоятельства, в которых проектная группа работает над созданием решения, обладают постоянной изменчивостью, следовательно, команда должна регулярно проводить переоценку выявленных рисков и постоянно следить за появлением новых. Управление рисками должно быть интегрировано в общий жизненный цикл проекта.

О положении дел в проекте нужно судить не по количеству рисков, связанных с его выполнением, а по степени проработанности процедуры их выявления, анализа и управления ими. Процесс управления рисками MSF определяет шесть логических шагов, посредством которых проектная группа управляет текущими рисками, разрабатывает и исполняет стратегии управления рисками и извлекает уроки из своего опыта для использования на уровне всего предприятия. Представм эти шаги:

- Выявление рисков (*risk identification*) — этап, позволяющий членам проектной группы вынести на обсуждение всей команды факты наличия рисков. Выявление рисков является начальной стадией процесса управления ими. Оно должно быть осуществлено как можно раньше, и к нему необходимо постоянно возвращаться на протяжении всего жизненного цикла проекта.
- Анализ рисков (*risk analysis*) — этап преобразования накопленных во время предыдущего шага оценок и данных в форму, позволяющую осуществить приоритезацию рисков. Приоритезация рисков (*risk prioritization*) позволяет проектной группе управлять наиболее важными из них, выделяя для этого необходимые ресурсы.
- Планирование рисков (*risk planning*) выполняется исходя из информации, полученной на этапе их анализа, и имеет целью выработку стратегий, планов и конкретных шагов. Календарное планирование рисков (*risk scheduling*) интегрирует эти планы в повседневный процесс управления проектом, обеспечивая непрерывность управления рисками. Эта стадия напрямую увязывает планирование рисков с планированием проекта в целом.
- Мониторинг рисков (*risk tracking*) выполняется для наблюдения за конкретными рисками и прогрессом в осуществлении составленных планов. Мониторингу должны быть подвергнуты сделанные оценки вероятности (*probability*) риска, его угрозы (*impact*), ожидаемая величина риска (*exposure*) и прочие факторы, способные повлиять на приоритет рисков. Наблюдению подвергаются также составленные планы, имеющиеся ресурсы и принятый календарный график.

- Корректирование ситуации (*risk control*) представляет собой процесс исполнения принятых в отношении рисков планов и контроля за ходом их исполнения. Этот процесс также включает в себя инициирование изменений всего проекта (*project change control requests*), если изменения в состоянии рисков либо в соответствующих планах влияют на прогнозируемый объем работы, требуемые ресурсы или сроки.
- Извлечение уроков (*risk learning*) формализует процесс усвоения накопленного за время работы над проектом опыта.

Необходимо отметить, что описанные этапы являются логическими шагами и не обязательно должны следовать друг за другом в строгом хронологическом порядке. Проектные группы могут циклически повторять шаги выявления-анализа-планирования по мере обнаружения дополнительных факторов, влияющих на проект. Некоторые возможные риски представлены в таблице 19.1.

Таблица 19.1 – Возможные риски

| № | Наименование риска | Комментарий |
|---|---|---|
| 1 | Не успеем сдать проект во время | Из-за неправильной организации работ затратим больше времени, чем заявлено в контракте |
| 2 | Не хватит квалификации персонала | При создании продукта используется новая технология, персонал с ней не работал и может не разобраться вовремя |
| 3 | Один из членов команды заболеет | Команда не многочисленна и отсутствие одного из членов команды ведет к задержке в работах |
| 4 | Заказчик изменит требования | В данный момент требования согласованы с заказчиком и утверждены, но в процессе работы заказчик может захотеть добавить в решение новую функциональность |
| 5 | Отключение электричества | Потеряем время, пока авария будет устраняться |
| 6 | Отключат доступ к Интернету | Ухудшится возможность быстрого получения необходимых сведений, пропадет электронная почта и другие средства коммуникации |
| 7 | Заказчик вовремя не оплатит счета | Задержка в покупке необходимого оборудования |
| 8 | Из-за необнаруженной вовремя ошибки система нанесет урон заказчику | На время устранения ошибки пассажиры не смогут заказывать билеты через систему. Потребуется «ручная» работа персонала. Компания потеряет возможных клиентов и часть прибыли |
| 9 | Нет требуемой аппаратуры | Не сможем адекватно развернуть систему |

продолжение на следующей странице

Таблица 19.1 – Возможные риски

| № | Наименование риска | Комментарий |
|---|---|---|
| 10 | Не сможем подобрать необходимые кадры | Придется совмещать роли |

В таблице 19.1 представлена лишь небольшая часть возможных рисков, однако можно заметить, что они происходят из самых разных областей и связаны с людьми (риск №3), с процессами (риски №1, №8, №10), с технологиями (риск №2), с внешними условиями (риски №5, №6), с заказчиком (риски №4, №7, №9). Помимо представленных риски могут иметь и другие источники.

Далее MSF рекомендует для каждого риска представить формулировку — выражение на естественном языке причинно-следственной связи между реально существующим фактором проекта и потенциально возможным, еще не случившимся событием или ситуацией. Первая часть формулировки риска называется условием (*condition*) и содержит описание существующего фактора или особенности проекта, которые, по мнению проектной группы, могут сделать результат проекта убыточным либо же сократить получаемую от проекта прибыль. Вторая часть формулировки риска называется (по)следствием (*consequence*). Она описывает ту нежелательную ситуацию, которой следует избежать. В качестве примера представим формулировки некоторых выявленных рисков, которые показаны в таблице 19.2.

Таблица 19.2 – Причины и возможный ущерб

| Первопричина | Условие | Последствие | Приносимый ущерб |
|---|---|---|---|
| Нехватка кадров | Один из членов команды заболеет | Функции заболевшего придется передать другому | Потери времени |
| Форс-мажор | Авария на подстанции, отключение электричества | Потеряем время, пока авария будет устраняться | После устранения аварии придется увеличить нагрузку, чтобы наверстать потери времени |
| Организация работы | Не сможем подобрать необходимые кадры | Участникам придется совмещать роли | Дополнительные трудозатраты, снижение качества продукта, увеличение времени разработки решения |

## Контрольные вопросы по главе 19

1) Что такое риски при разработке ПО?
2) В чем заключается цель управления рисками?
3) Укажите основные концепции понимания рисками в MSF?
4) Что такое концепция «риск» согласно концепции MSF?
5) Что такое концепция «Выявление рисков нужно всячески одобрять» согласно концепции MSF?
6) Что такое концепция «Оценка рисков должна вестись постоянно» согласно концепции MSF?
7) Сколько и каких логических шагов определяет процесс управления рисками MSF?
8) Что означают шаги «Выявление рисков» и «Анализ рисков» управления рисками согласно MSF?
9) Что означают шаги «Планирование рисков» и «Мониторинг рисков» управления рисками согласно MSF?
10) Приведите основные риски, возникающие при разработке ПО?

# Глава 20

# CMM — СТАНДАРТ КАЧЕСТВА ПО

CMM SW — Capability Maturity Model for Software — американский стандарт в области качества ПО, разработанный SEI по заказу министерства обороны США. Этот стандарт появился в 1993 году и быстро получил широкое международное признание. Главным образом потому, что этот стандарт предназначен только для разработки ПО и по отношению к остальным стандартам управление качеством для разработки ПО в нем прописано достаточно подробно и детально.

Исследователи SEI пошли достаточно простым путем, т. к. оценку зрелости организаций они решили проводить на основе анализа выполняемых этой организацией процессов по разработке ПО. При этом считалось, что организация является тем более зрелой (более предсказуемой), чем более установленными являются применяемые процессы. Первые исследования организаций с этих позиций показали, что организации находятся на разных уровнях зрелости — была проведена классификация этих уровней, разработаны методы оценки уровня организации и предложены способы повышения уровня зрелости организации. Все это вместе и составляет модель технологической зрелости организации, или модель зрелости ее технологических процессов.

*В CMM дается следующее определение:* ***Модель технологической зрелости*** *— это описание стадий эволюции, которые проходят организации-разработчики по мере того, как они (организации) определяют, реализуют, измеряют, контролируют и совершенствуют процессы создания ПО.*

Модель помогает выбрать адекватную стратегию усовершенствования процессов, предоставляет методическую основу для определения текущего уровня их совершенства и выявления проблем, критичных для качества разрабатываемого ПО.

Основу модели CMM составляют следующие фундаментальные понятия:

- Process (технология, технологический процесс, процесс) — последовательность шагов (действий), предпринимаемых с заданной целью. Более точно, процесс определяется так: производственный процесс — набор операций, методов, практик и преобразований, используемых разработчиками для создания и сопровождения ПО и связанных с ним продуктов (например, планов проекта, проектных документов, кодов, сценариев тестирования и руководств пользователя).
- Process Capability (продуктивность, совершенство технологии/процесса) — диапазон результатов, которые можно ожидать от организации, соблюдающей данный технологический процесс. Это понятие имеет отношение к будущим проектам, но базируется на фактических характеристиках технологии, достигнутых на предыдущих проектах.
- Process Performance (производительность технологии/процесса) — фактические результаты, достигнутые организацией, соблюдающей данную технологию/процесс. Это понятие ассоциируется с уже выполненными проектами.
- Таким образом, производительность фокусируется на достигаемых результатах, в то время как его продуктивность опирается на ожидаемые результаты.
- Process Maturity (зрелость технологии) — степень определенности, управляемости, наблюдаемости, контролируемости и эффективности процесса, технологии. Фактически, это индикатор полноты технологии и степени последовательности (настойчивости) организации в ее применении на всех проектах. Зрелость определяет потенциал дальнейшего роста совершенства технологии/процесса.

Разработчики модели CMM (SEI) определили пять уровней технологической зрелости, по которым заказчики могут оценивать потенциальных поставщиков (претендентов на получение контракта), а поставщики могут совершенствовать процессы создания ПО. Каждому из уровней технологической зрелости внутри модели CMM дано следующее краткое определение:

- Начальный (Initial). Технология разработки ПО характеризуется как произвольная (импровизированная), в некоторых случаях — даже хаотическая. Лишь некоторые процессы определены, успех всецело зависит от усилий отдельных сотрудников.
- Повторяемый (Repeatable). Базовые процессы управления проектом ПО установлены для отслеживания стоимости, графика и функциональности выходного продукта. Необходимая дисциплина соблюдения установленных процессов имеет место и обеспечивает возможность повторения успеха предыдущих проектов в той же прикладной области.
- Определенный (Defined). Управленческие и инженерные процессы задокументированы, стандартизованы и интегрированы в унифицированную для всей организации технологию создания ПО. Каждый проект использует утвержденную, адаптированную к особенностям данного проекта версию этой технологии.
- Управляемый (Managed). Детальные метрики (объективные данные) о качестве исполнения процессов и выходной продукции собираются и накап-

ливаются. Управление процессами и выходной продукцией осуществляется по количественным оценкам.

- Оптимизируемый (Optimized). Совершенствование технологии создания ПО осуществляется непрерывно на основе количественной обратной связи от процессов и плотного внедрения инновационных идей.

Модель зрелости CMM является гибкой — она не содержит четких и конкретных (формализованных) указаний на этот счет, а дает некоторую схему поиска ответов на поставленные вопросы и рекомендации по ее использованию. По мнению разработчиков, это расширяет применимость модели. Структура (схема) модели CMM содержит следующие основные элементы:

- *Группы ключевых процессов.* Каждый уровень зрелости содержит описание группы ключевых процессов, которые должны выполняться на этом уровне.
- *Цели.* Для каждого ключевого процесса определены цели, которые нужно достигнуть для перехода на следующий уровень. Цели (целевые установки):
  - служат критерием эффективной реализации группы ключевых процессов в организации, выражают объем, границы и смысл каждой группы ключевых процессов. После того, как цели будут реализованы на постоянной основе для всех проектов, можно будет сказать, что организация установила уровень продуктивности своего производственного процесса, характеризующийся данной группой ключевых процессов. Кроме того, определяется блок связанных работ, после выполнения которых достигаются цели, и круг проблем, которые необходимо решить для достижения следующего уровня зрелости.
- *Разделы*. Описания ключевых процессов организованы по разделам, которые представляют собой атрибуты, указывающие, являются ли эффективными, повторяемыми и устойчивыми реализация и установление групп ключевых процессов. Ниже перечислены пять основных разделов:
  - *Обязательства по выполнению*. Описывают действия, которые должна выполнить организация, чтобы обеспечить установление и стабильность процесса. Обязательства по выполнению обычно касаются установления организационных политик и поддержки со стороны высшего руководства.
  - *Необходимые предпосылки*. Описывают предварительные условия, которые должны выполняться в проекте или организации для компетентного внедрения производственного процесса, обычно касаются ресурсов, организационных структур и требуемого обучения.
  - *Выполняемые операции.* В разделе «Выполняемые операции» описаны роли и процедуры, необходимые для внедрения группы ключевых процессов. Выполняемые операции обычно включают в себя создание планов и реализацию процедур, выполнение и отслеживание работ, а также, по мере необходимости, выполнение корректирующих действий.

  - *Измерения и анализ*. Раздел «Измерения и анализ» описывает, что необходимо для измерения процесса и анализа результатов измерений. В этом разделе обычно приводятся примеры измерений, с помощью которых можно определить статус и эффективность выполняемых операций.
  - *Проверка внедрения*. В разделе «Проверка внедрения» описываются шаги, позволяющие убедиться в том, что операции выполняются в соответствии с установленным процессом. В этот раздел обычно входят проверки и аудиты со стороны руководства и работы по обеспечению качества ПО.
- *Ключевые практики.* Описание каждого раздела ключевых процессов выражается ключевыми практиками и подпрактиками, выполнение которых способствует достижению целей группы. Ключевые практики описывают инфраструктуру и операции, которые дают наибольший вклад в эффективное внедрение и установление группы ключевых процессов.

В 2003 году американский Институт программной инженерии *(SEI)* опубликовал новую комплексную модель *CMMI*, уточняющую и совершенствующую предшествовавшие модели *CMM*, а также частично учитывающую основные требования существующих международных стандартов в области менеджмента программных средств. Внедрение этой модели акцентировано на улучшении процессов управления проектами ПС, обеспечении их высокого качества и конкурентоспособности, с основной целью — сделать процессы проектов *управляемыми,* а результаты — *предсказуемыми.* Значительное внимание в *CMMI* уделяется процессам разработки и учету итераций при изменении требований заказчиков, их прослеживанию к функциям, компонентам, тестам и документам проекта. Концепцию, определяющую функциональную пригодность и качество продукта, рекомендуется сопровождать проверками возможных сценариев событий при его применении.

## Контрольные вопросы по главе 20

1) Что такое стандарт качества CMM?
2) Какие понятия положены в основу стандарта качества CMM?
3) Что означает понятие «процесс» и «продуктивность» стандарта качества CMM?
4) Что означают понятия «производительность технологии/процесса» и «зрелость технологии» стандарта качества CMM?
5) Приведите уровни технологической зрелости стандарта качества CMM?
6) Что означают уровни «Начальный» и «Повторяемый» технологической зрелости стандарта качества CMM?
7) Что означают уровни «Определенный» и «Управляемый» технологической зрелости стандарта качества CMM?
8) Что означает уровень «Оптимизируемый» технологической зрелости стандарта качества CMM?

# Глава 21

# РАЗРАБОТКА ТРЕБОВАНИЙ К ПО

Разработчику хочется сделать программу высокотехнологичной и полезной для покупателей. Продавцу же нужна простая и полностью готовая к эксплуатации программная система, а для покупателя важны удобство и функциональные возможности. Напряженность между этими тремя сторонами с их различными целями, ограничениями может привести к несогласованности бизнес-требований. Бизнес-требования определяют набор бизнес-задач (вариантов использования), которые позволяют выполнять приложение *(ширина приложения),* и *глубину уровня,* до которого реализуется каждый вариант использования.

Таблица 21.1 – Реализация приемов формулирования требований

| **Влияние** | **Сложность** | | |
|---|---|---|---|
| | **высокая** | **средняя** | **низкая** |
| Сильное | 1. Определите процесс формулирования требований<br>2. Планируйте на основании требований<br>3. Пересматривайте обязательства | 1. Определите варианты использования<br>2. Укажите атрибуты качества<br>3. Определите приоритеты требований<br>4. Используйте шаблон спецификации требований к ПО<br>5. Определите процесс управления изменениями<br>6. Создайте совет | 1. Обучите разработчиков основам предметной области<br>2. Определите образ и границы проекта<br>3. Определите классы пользователей<br>4. Нарисуйте контекстную диаграмму<br>5. Определите источники требований<br>6. Определите базовую версию и управ- |
| продолжение на следующей странице | | | |

Таблица 21.1 – Реализация приемов формулирования требований

| **Влияние** | **Сложность** | | |
|---|---|---|---|
| | **высокая** | **средняя** | **низкая** |
| Сильное | | управления изменениями<br>7. Изучите документы с требованиями<br>8. Распределите требования по подсистемам<br>9. Задокументируйте бизнес-правила | ляйте версиями требований |
| Среднее | 1. Ознакомьте представителей пользователей и менеджеров с требованиями<br>2. Моделируйте требования<br>3. Управляйте рисками, связанными с требованиями<br>4. Используйте утилиту управления требованиями<br>5. Создайте матрицу связей требований<br>6. Проводите семинары для уточнения требований | 1. Обучите аналитиков требований<br>2. Выберите ярых сторонников продукта<br>3. Создайте фокус-группы<br>4. Создайте прототипы<br>5. Определите критерии приемлемости<br>6. Анализируйте влияние изменений<br>7. Выберите соответствующий цикл | 1. Проанализируйте осуществимость<br>2. Создайте словарь бизнес-терминов<br>3. Создайте словарь данных<br>4. Наблюдайте за пользователями на рабочих местах<br>5. Определите системные события и реакцию на них<br>6. Задайте каждому требованию уникальный идентификатор<br>7. Протестируйте требования<br>8. Отслеживайте состояние требований<br>9. Извлекайте уроки из полученного опыта |
| Слабое | 1. Используйте требования повторно<br>2. Воспользуйтесь технологией развертывания функций качества<br>3. Оцените изменяемость требований | 1. Ведите журнал изменений<br>2. Отслеживайте объем работ по реализации требований | 1. Изучите отчеты о проблемах |

## Контрольные вопросы по главе 21

1) Какие требования выдвигаются при разработке ПО?
2) Какие уровни сложности ПО существуют?
3) Что влияет на разработку ПО различной сложности?

# Глава 22

# О ПРИМЕНИМОСТИ МЕТОДОВ ПРОГРАММНОЙ ИНЖЕНЕРИИ К ГРУППОВОМУ ПРОЕКТНОМУ ОБУЧЕНИЮ

Наиболее применяемым подходом к разработке программного обеспечения являются идеи MSF. Приведем кратко описание основных идей подхода к формированию проектной группы и основных функций группы во время жизненного цикла программы.

Методология MSF считает, что успешная работа команды над проектом существенным образом зависит от ее структуры и распределения зон ответственности ролевых групп (более подробно о составе проектной группы далее) внутри команды. Построение команды в MSF соответствует ряду ключевых концепций (*key concepts*), часть которых кажутся самоочевидными, другие чем-то сродни «ноу-хау».

К самоочевидным можно отнести:

- Концентрацию на нуждах заказчика (*customer-focused mindset*) — главный приоритет любой хорошо работающей проектной группы. Означает обязательное понимание бизнес-задач заказчика и стремление к их решению со стороны команды. Не менее важным является активное участие заказчика в проектировании решения и получение его отзывов в ходе процесса разработки.
- Нацеленность на конечный результат (*product mindset*) — каждый участник проектной группы должен рассматривать собственную работу в качестве самостоятельного проекта или же вклада в какой-либо больший проект. Установка на конечный продукт означает, что получению конечного результата проекта уделяется больше внимания, чем процессу его достижения. Из этого не следует, что сам процесс может быть плох или непродуман — просто он существует для получения конечной цели, а не ради себя самого.

- Установка на отсутствие дефектов (*zero-defect mindset*) — это стремление к высочайшему уровню качества. Она означает, что цель команды — выполнение своей работы с максимально возможным качеством, в идеале таким образом, что если от команды потребуют поставить результат завтра, она будет способна поставить что-то работающее. В успешной команде каждый сотрудник чувствует ответственность за качество продукта. Она не может быть делегирована одним членом команды другому или же от одной ролевой группы другой.

Концепции, которые в определенном смысле можно отнести к «ноу-хау» методологии MSF:

«Проектная группа — команда равных» (*teem of peers*). Концепция означает равноправное положение каждой из ролей в команде. Чтобы достичь успеха в рамках команды равных, каждый из ее членов, независимо от роли, должен нести ответственность за качество продукта, понимать интересы заказчика и сущность решаемой бизнес-задачи. В то же время, принятие решения методом консенсуса между ролями не тождественно принятию решения методом консенсуса между сотрудниками. Каждая ролевая группа требует определенной организационной иерархии для распределения работы и управления ее ресурсами.

Методология MSF основана на постулате о качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают модель проектной группы. В то время как за успех проекта ответственна вся команда, каждая из ее ролевых групп, определяемых моделью, ассоциирована с одной из целей и работает над ее достижением.

MSF for Agile Software Development выделяет 7 ролевых групп:

- Управление программой (program management).
- Архитектура продукта (architecture).
- Разработка (development).
- Тестирование (test).
- Управление выпуском (release operations).
- Удовлетворение потребителя (user experience).
- Управление продуктом (product management)

и 6 ролей:

- Менеджер проекта (project manager) — ролевая группа «Управление программой».
- Архитектор (archrect) — ролевая группа «Архитектура».
- Разработчик (developer) — ролевая группа« Разработка».
- Тестер (tester) — ролевая группа «Тестирование».
- Релиз-менеджер (release manager) — ролевая группа «Управление выпуском».
- Бизнес-аналитик (business analyst) — ролевые группы «Управление продуктом».

Ролевой кластер — согласованность работы команды, соблюдение временных ограничений. Классическая модель проектной группы MSF, сочетающей и ролевые группы, и роли, изображена на рис. 22.1.

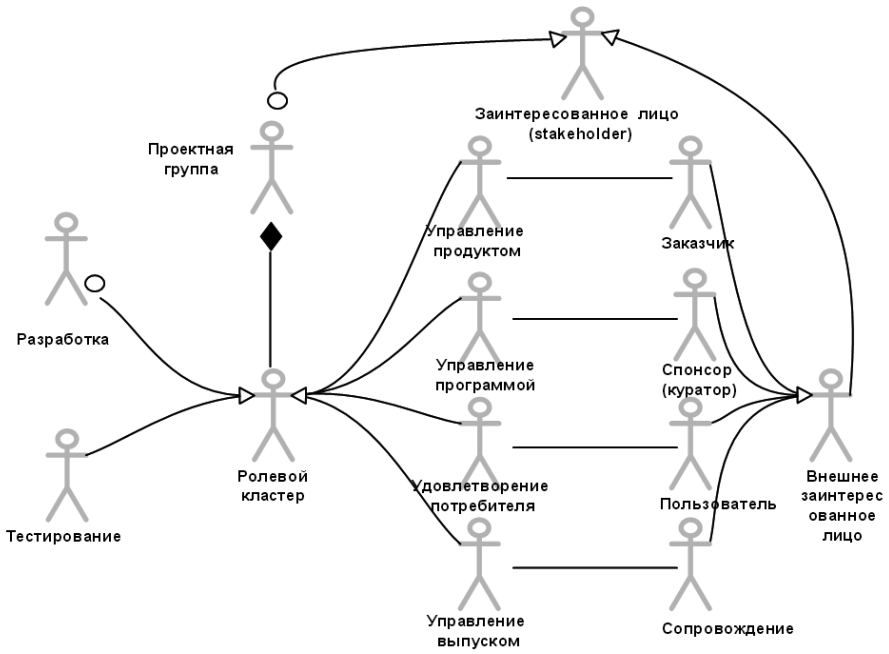



Рис. 22.1 – Модель проектной группы MSF

Особенностью модели процессов MSF является подход, основанный на фазах и вехах. MSF вводит два типа вех: главные (major) и промежуточные (interim). Они имеют следующие особенности:

- Главные вехи служат точками перехода от одной фазы к другой. Они также определяют изменения в текущих задачах ролевых кластеров.
- В MSF используются обобщенные главные вехи, большинство из которых применимо к любому типу IT-проектов.
- Промежуточные вехи показывают достижение в ходе проекта определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки.
- Промежуточные вехи могут варьироваться от проекта к проекту. MSF рекомендует использовать определенный набор промежуточных вех, но на практике проектная группа может сама устанавливать их в соответствии с особенностями своей работы.

Главные вехи — это моменты жизненного цикла проекта, когда полученные на той или иной фазе результаты синхронизируются членами проектной группы друг с другом и с ожиданиями заказчика. В этот момент заказчиком, заинтересованными сторонами и проектной группой производится формальный анализ достигнутого прогресса. Успешное прохождение главной вехи знаменует согласие проектной группы и заказчика продолжать далее работу над проектом. Вехи позволяют заказчику и проектной группе проверить соответствие рамок проекта потенциально изменяющимся требованиям, и если необходимо, скорректировать их, а также отреагировать на возникающие риски.

Хотя ролевой кластер организует работу над проектом в целом, на каждой из фаз определенные ролевые кластеры имеют ведущее значение. Объем работы,

выполняемой различными ролевыми кластерами, меняется в процессе перехода проекта из одной фазы в другую. Использование проектных вех помогает должным образом организовать эти переходные процессы.

Между ролевыми кластерами и главными вехами существует определенное соответствие. Оно указывает, какие именно роли несут первоочередную ответственность за достижение каждой из вех. Переход от одной фазы к другой включает в себя также перенос основной ответственности от одних ролевых кластеров к другим.

Нижеследующая таблица показывает, какие роли являются первостепенными в достижении каждой из главных вех. Однако следует отметить, что ведущее положение одних ролей никоим образом не исключает участие в работе остальных ролевых кластеров.

Таблица 22.1 – Соответствие между главными вехами и ведущими ролевыми кластерами

| Веха | Ведущие ролевые кластеры |
|---|---|
| Концепция утверждена | Управление продуктом |
| Планы проекта утверждены | Управление программой |
| Разработка завершена | Разработка. Удовлетворение потребителя |
| Готовность решения утверждена | Тестирование. Управление выпуском |
| Внедрение завершено | Управление выпуском |

## Основные задачи проектной группы на фазе выработки концепции

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

Следующая таблица описывает основные задачи и сферы ответственности каждого из ролевых кластеров проектной группы во время фазы выработки концепции.

Таблица 22.2 – Задачи и среды ответственности ролевых кластеров проектной группы во время фазы выработки концепции

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

## Основные задачи проектной группы на фазе планирования

На фазе планирования (planning) производится основная работа по составлению планов проекта. Она включает в себя подготовку проектной группой функциональной спецификации, разработку дизайнов, подготовку рабочих планов, оценку проектных затрат и сроков разработки различных составляющих проекта.

В начале фазы планирования проектная группа анализирует и документирует проектные требования. Они разделяются на четыре общие категории: бизнес-требования (business requirements), потребительские требования (user requirements), эксплуатационные требования (operational requirements) и системные требования, относящиеся к решению в целом (system requirements). В ходе проектирования решения и создания его функциональной спецификации необходимо следить за *соответствием (traceability)* между имеющимися требованиями и проектируемой функциональностью. Это соответствие не обязательно будет взаимооднозначным. Оно служит одним из способов контроля корректности дизайна и его пригодности для достижения поставленных перед решением целей.

Следующая таблица описывает основные задачи и сферы ответственности каждого из ролевых кластеров проектной группы во время фазы планирования.

Таблица 22.3 – Задачи и сферы ответственности ролевых кластеров проектной группы во время фазы планирования

| Ролевой кластер | Фокус |
|---|---|
| Управление продуктом | Концептуальный дизайн; анализ бизнес-требований; коммуникационный план |
| Управление программой | Концептуальный и логический дизайн; функциональная спецификация; сводный план и сводный календарный график проекта; бюджет |
| Разработка | Оценка технологий; логический и физический дизайн; план и календарный график разработки; смета разработки (development estimates) |
| Удовлетворение потребителя | Сценарии/примеры использования, пользовательские требования, требования локализации и общедоступности (accessibility); пользовательская документация (план обучения), график тестирования, удобства эксплуатации; обучение |
| Тестирование | Оценка дизайна; требования тестирования; план и календарный график тестирования |
| Управление выпуском | Оценка дизайна; эксплуатационные требования; план и календарный график пилотного и окончательного внедрения |

# Основные задачи проектной группы на фазе разработки

На фазе разработки проектная группа фокусируется на создании компонент решения (включая как документацию, так и программный код). Однако некоторая часть этой работы может продолжаться также на фазе стабилизации, если такая необходимость выявлена в процессе тестирования. Данная фаза также включает в себя разработку инфраструктуры. Следует обратить внимание, что активность проектной команды на этом этапе не ограничивается написанием разработчиками кода — все ролевые кластеры принимают деятельное участие в создании и тестировании решения.

Следующая таблица описывает основные задачи и сферы ответственности каждого из ролевых кластеров проектной группы во время фазы разработки.

Таблица 22.4 – Задачи и сферы ответственности ролевых кластеров проектной группы во время фазы разработки

| Ролевой кластер | Фокус |
|---|---|
| Управление продуктом | Ожидания заказчика |
| Управление программой | Управление функциональной спецификацией; мониторинг проекта; доработка планов |

продолжение на следующей странице

Таблица 22.4 – Задачи и сферы ответственности ролевых кластеров проектной группы во время фазы разработки

| Ролевой кластер | Фокус |
|---|---|
| Разработка | Разработка программного кода и инфраструктуры; документирование конфигураций |
| Удовлетворение потребителя | Обучение; доработка плана обучения; тестирование удобства эксплуатации (usability testing); графический дизайн |
| Тестирование | Функциональное тестирование; выявление проблем; тестирование документации; доработка плана тестирования |
| Управление выпуском | Чеклисты развертывания (rollout checklists); доработка планов внедрения (включая пилотное внедрение); чеклисты подготовки к внедрению (site preparation checklists) |

## Основные задачи проектной группы на фазе стабилизации

Во время фазы стабилизации производится тестирование разработанного решения. При этом внимание фокусируется на его эксплуатации в реалистичной модели производственной среды. Проектная группа занимается приоритезацией и устранением ошибок, а также подготовкой решения к выпуску.

Обычно в начале фазы стабилизации скорость выявления ошибок командой тестирования превосходит скорость, с которой эти ошибки могут устраняться командой разработчиков. Невозможно предсказать, сколько ошибок будет найдено и как много времени понадобится на их устранение. Однако существует два статистических признака, помогающих проектной группе оценить уровень стабилизации решения. Это точка конвергенции (bug convergence) и точка достижения нуля ошибок (zero bug bounce).

В точке конвергенции (bug convergence) становится заметен существенный прогресс в устранении ошибок, то есть скорость устранения ошибок начинает превосходить скорость их обнаружения. Точка достижения нуля (zero-bug bounce) — это момент, когда впервые все выявленные ошибки оказываются устраненными. Существенную роль играет тщательный анализ ошибок, поскольку устранение всякой из них содержит риск внесения новых ошибок. Точка достижения нуля ясно показывает, что проектная группа приближается к созданию стабильной версии-кандидата (release candidate).

Следующая таблица описывает основные задачи и сферы ответственности каждого из ролевых кластеров проектной группы во время фазы стабилизации.

Таблица 22.5 – Задачи и сферы ответственности каждого ролевого кластера проектной группы во время фазы стабилизации

| Ролевой кластер | Фокус |
|---|---|
| Управление продуктом | Исполнение коммуникационного плана; планирование премьеры продукта |
| Управление программой | Мониторинг проекта; приоритезация ошибок |
| Разработка | Устранение ошибок; оптимизация программного кода |
| Удовлетворение потребителя | Доработка эксплуатационных руководств; учебные материалы |
| Тестирование | Тестирование; сообщение об ошибках и их статусе; тестирование конфигурации |
| Управление выпуском | Развертывание и поддержка пилотного внедрения; планирование внедрения; обучение персонала сопровождения |

## Основные задачи проектной группы на фазе внедрения

Во время этой фазы проектная группа внедряет технологии и компоненты решения, стабилизирует внедренное решение, передает работу персоналу поддержки и сопровождения и получает со стороны заказчика окончательное одобрение результатов проекта. По завершению внедрения проектная группа производит анализ выполненной работы и удовлетворенности заказчика.

Следующая таблица описывает основные задачи и сферы ответственности каждого из ролевых кластеров проектной группы во время фазы внедрения.

Таблица 22.6 – Задачи и сферы ответственности каждого ролевого кластера проектной группы во время фазы внедрения

| Ролевой кластер | Фокус |
|---|---|
| Управление продуктом | Получение отзывов и оценок заказчика; акт о приеме выполненной работы |
| Управление программой | Сопоставление рамок проекта с поставленным решением; управление стабилизацией |
| Разработка | Разрешение проблем; поддержка эскалации |
| Удовлетворение потребителя | Обучение; управление календарным графиком обучения |
| Тестирование | Тестирование производительности |
| Управление выпуском | Управление внедрением; одобрение изменений |

В том случае, когда группа разработчиков небольшая, один человек может выполнять в проектной группе несколько ролей. В этом случае MSF предлагает рекомендации по возможному объединению ролей. Рассмотрим их в виде таблицы 22.7.

Таблица 22.7 – Возможности по объединению ролей внутри группы

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| менеджер проекта, релиз-менеджер (тестировщик) | | - | + | + | * | * | * |
| архитектор и разработчик | - | | - | - | + | + | * |
| бизнес-аналитик и тестер | + | - | | - | * | * | + |
| разработчик и подготовка документации | + | - | - | | - | - | - |
| разработчик и подготовка документации | * | + | * | - | | + | + |
| разработчик и подготовка документации | * | + | * | - | + | | * |
| разработчик и внедрение | * | * | + | - | + | * | |

В таблице знак «+» означает — ДА, «-» означает НЕТ и «*» означает НЕЖЕЛАТЕЛЬНО.

Следуя правилам, установленным в вышепредставленной таблице, примерный состав группы проектного обучения по разработке программного продукта может быть таким:

- участник 1 — менеджер проекта, релиз-менеджер (тестировщик),
- участник 2 — архитектор и разработчик,
- участник 3 — бизнес-аналитик и тестер,
- участник 4 — разработчик и подготовка документации,
- участник 5 — разработчик и подготовка документации,
- участник 6 — разработчик и внедрение,
- участник 7 — разработчик и внедрение.

Таким образом, видно, что группа проектного обучения может состоять примерно из семи человек, каждый из которых имеет несколько ролей, которые покрывают весь список возможных ситуаций при разработке программы, от разработки до внедрения.

# ЗАКЛЮЧЕНИЕ

В конце 60-х — начале 70-х годов прошлого века произошло событие, которое вошло в историю как первый кризис программирования. Событие состояло в том, что стоимость программного обеспечения стала приближаться к стоимости аппаратуры («железа»), а динамика роста этих стоимостей позволяла прогнозировать, что к середине 90-годов все человечество будет заниматься разработкой программ для компьютеров. Тогда и заговорили о программной инженерии (или технологии программирования, как это называлось в России) как о некоторой дисциплине, целью которой является сокращение стоимости программ. С тех пор программная инженерия прошла достаточно бурное развитие. Этапы развития программной инженерии можно выделять поразному. Каждый этап связан с появлением (или осознанием) очередной проблемы и нахождением путей и способов решения этой проблемы. В данном пособии, весьма кратко рассмотрены основные положения такого направления, как програмная инженерия. Хочется верить, что студенты прочитают пособие и разберутся в методах, составляющих основу подходов к проектированию и разработке программных продуктов.

# ЛИТЕРАТУРА


[1] Модель зрелости процессов разработки программного обеспечения / М. Паулк [и др.]. — Capability Maturity Model for Software (CMM) Интерфейс-Пресс, 2002. — 256 с.

[2] Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 CMM) : пер. с англ. А. С. Агапова [и др.]. — М. : Книга и бизнес, 2001. — 348 с.

[3] Шафер Д. Управление программными проектами: достижение оптимального качества при минимуме затрат : пер. с англ. / Д. Шафер, Р. Фатрел, Л. Шафер. — М. : Вильямс, 2003. — 1136 с.

[4] ГОСТ Р ИСО/МЭК 12207–99. Процессы жизненного цикла программных средств.

[5] Константин Л. Человеческий фактор в программировании : пер. с англ. / Л. Константин. — СПб. : Символ-Плюс, 2004. — 384 с.

[6] Салливан Эд. Время — деньги. Создание команды разработчиков программного обеспечения : пер. с англ. / Эд. Салливан. — М. : Русская редакция, 2002. — 364 с.

[7] Соммервилл И. Инженерия программного обеспечения / И. Соммервилл. — М. : Вильямс, 2002–543 с.

[8] Якобсон А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. — СПб. : Питер, 2002. — 455 с.

[9] Брауде Э. Дж. Технология разработки программного обеспечения / Э. Дж. Брауде. — СПб. : Питер, 2004. — 399 с.

[10] Леффингуэлл Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход / Д. Леффингуэлл, Д. Уидриг. — М. : Вильямс, 2002. — 435 с.

[11] Липаев В. В. Программная инженерия. Методологические основы / В. В. Липаев. — М. : Наука, 2006. — 609 с.

[12] Жоголев Е. А. Лекции по технологии программирования : учеб. пособие / Е. А. Жоголев. — М. : Издательский отдел факультета ВМиК МГУ, 2001. — 211 с.

[13] Характеристики качества программного обеспечения / Боэм Б. [и др.]. — М. : Мир, 1991. — 325 с.

[14] Ройс У. Управление проектами по созданию программного обеспечения / У. Ройс. — М. : Лори, 2002. — 524 с.

[15] Бек К. Экстремальное программировнаие / К. Бек. — СПб. : Питер, 2002. — 128 с.

[16] Архитектура корпоративных программных приложений / М. Фаулер [и др.]. — М. : Вильямс, 2004. — 33 с.

[17] Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. — СПб. : Питер-ДМК, 2001. — 432 с.

[18] Р. Дж. Торрес. Практическое руководство по проектированию и разработке пользовательского интерфейса / Р. Дж. Торрес. — М. : Вильямс, 2002. — 288 с.

[19] http://www.microsoft.com/msf

[20] http://www.scribd.com/doc/7331774/MSF-Team-Model-v3

[21] www.itlab.unn.ru/archive/seminar/SE

[22] ru.wikipedia.org/wiki/Microsoft_Solutions_Framework

[23] www.swebok.org/

[24] http://swebok.sorlik.ru/

# Приложение А

# ХАРАКТЕРИСТИКА СТАНДАРТОВ РАЗРАБОТКИ АВТОМАТИЗИРОВАННЫХ СИСТЕМ (АС)

## А.1 Характеристика ГОСТ 34.601--90 для разработки ПС

Данный стандарт определяет стадии и этапы разработки автоматизированных систем (АС). В зависимости от конкретных условий стадии и этапы могут объединяться друг с другом. В стандарте определено восемь этапов разработки АС, на каждом из которых формируются документы, описываемые ниже.

*1 этап. Формирование требований к автоматизированной системе (АС).* Проводится обследование объекта и обоснование необходимости создания АС. Формулируются требования пользователя к АС, оформляются отчет о выполненной работе. Выясняется документооборот, формы начальных и исходящих документов, методики расчета отдельных показателей. Отчет об обследовании создается в произвольной форме, является основой разработки технического проекта АС, в приложениях к отчету приводятся формы документов и методики расчета экономических показателей АС.

Сформулированные требования к системе — заявка на разработку технического задания.

*2 этап. Разработка концепции АС.* Определяются пути и оценки возможностей реализации требований пользователя, а также методы, которые будут положены в основу расчетов, и подходы к решению конкретных задач системы. Заканчивается этап созданием и утверждением отчета о научно-исследовательской работе, в котором дается оценка необходимых для реализации ресурсов, вариантов разработки и оценки качества АС.

*3 этап. Разработка технического задания (ТЗ).* ТЗ — это основной документ, который определяет требования и порядок создания (развития или модернизации)

АС. На основании ТЗ ведется разработка АС, ее прием во время ввода в действие. Дополнительно могут быть разработаны ТЗ на отдельные части АС.

*4 этап. Разработка эскизного проекта.* На основе проектных решений ко всей системе или ее частям определяется перечень задач, которые будут выполняться в системе, концепция информационной базы, функции и параметры основных программных средств. Для каждой задачи приводятся согласованные с заказчиком формы первичных и исходящих документов, структура информационных массивов или их перечень, основные алгоритмы обработки информации.

*5 этап. Разработка технического проекта (ТП).* В ТП дается описание разработанных проектных решений для системы и ее частей, документации на АС и комплектации АС или технических требований на нее, задач проектирования смежных частей проекта, для которых проектные решения определяют организационную структуру, функции персонала в АС, структуру технических средств, языки программирования, СУБД, общие характеристики ПО, система классификации и кодирование, а также варианты информационной базы.

*6 этап. Разработка рабочей документации.* На этом этапе создаются проектные документы, которые определяются государственными стандартами и включают: постановки задач, алгоритмы их решения, организацию информационного, технического и программного обеспечения. Эти проектные документы могут оформляться как отдельные документы, а могут входить в технический проект как отдельные разделы. К документам рабочего проекта относятся: общее описание системы, описание технологического процесса обработки информации, инструкции по выполнению отдельных операций технологического процесса, руководство пользователя, описание программ и т. п.

*7 этап. Введение АС в эксплуатацию.* При создании рабочего проекта проводится разработка и отладка программ или адаптация готовых программ, которые разрабатывались для других объектов, их описание или паспорт. На этапе ввода в эксплуатацию выполняется: подготовка объекта к вводу в эксплуатацию, комплектация АС, установка технических и программных средств, выполнение монтажных работ и проведение приемочных испытаний. Готовится приказ об изменениях в структуре объекта, документообороте, а также о распределении обязанностей персонала при переходе на новую технологию обработки информации. Параллельно с подготовкой персонала ведутся работы по установке технических и программных средств, а также разрабатываются средства охраны и защиты АС. Предварительные испытания системы выполняет разработчик на основе контрольного примера или реальных данных. По результатам опытной эксплуатации программного обеспечения могут вноситься изменения, которые могут повлечь за собой доработку технического проекта системы. После завершения опытной эксплуатации проводятся приемочные испытания соответственно ТЗ специально созданной комиссией. В результате создается акт введения системы в эксплуатацию.

*8 этап. Сопровождение АС.* Во время сопровождения устраняются недостатки, которые обнаружены во время эксплуатации, и создается акт о выполненных работах. По мере внесения изменений в рабочую документацию могут вноситься изменения и в технический проект, как правило, самими разработчиками. АС может создаваться на основе готовых типовых программных средств, которые ориентированы на предметную область этой АС. Программные средства могут продаваться

разработчиком или его представителем. В этом случае работы для заказчика выполняются в одну стадию — введение в эксплуатацию АС. Проводится экспертиза, принимается решение о закупке необходимых программных средств. Кроме того, готовится приказ об изменении технологии работы на отдельных участках проекта АС и определяются ответственные за внедрение новой технологии. Разработчик передает заказчику системы рабочую документацию на АС или на ее отдельные части. Все перечисленные работы создаются по договору между заказчиком и разработчиком.

## А.2 Стандарт разработки документации на ПС — ГОСТ 34.201–89

Стандарт 34.201–89 («Информационная технология. Виды, комплектность и типы документов при создании АС») определяет набор разных видов документов и их комплектность для разрабатываемой АС. К документам относятся: отчеты об обследовании предметной области, результаты научно-исследовательской работы, техническое задание, эскизный проект, технический проект и рабочий проект. Отчеты об обследовании и научно-исследовательской работе, а также эскизный проект АС создаются в произвольной форме, их структура и содержание согласуются с заказчиком и разработчиком системы. Содержание и структуру технического задания, технического и рабочего проектов определяют соответствующие государственные стандарты, например ГОСТ 34.601–90. Техническое задание для АС — это основной документ, который определяет требования и порядок его создания или модернизации и может содержать такие разделы:

1) Общие сведения.
2) Назначение и цель создания системы.
3) Характеристика объектов автоматизации.
4) Требования к системе.
5) Состав и содержание работ по созданию системы.
6) Порядок контроля и приемки системы.
7) Требования к документации.
8) Источники разработки.

В техническое задание разрешается вносить некоторые новые разделы или их объединять и детализировать отдельно.

*1. Общие сведения.* Раздел знакомит со структурой организации заказчика и разработчика, определяет источники финансирования разработки, сроки начала и окончания работ, порядок оформления результатов проектных работ.

*2. Назначение и цель создания системы.* Раздел содержит описание целей создаваемой АС, ее назначение и возможности ее автоматизации с применением средств вычислительной техники. Дается обоснование необходимости создания АС и решения о выделении необходимого финансирования.

*3. Характеристика объекта автоматизации.* Раздел содержит важнейшие сведения об объекте (или ссылку на документы, где такие сведения можно найти). Например, информация о наличии вычислительной техники, размещение подразделов, основные их функции и т. п.

*4. Требования к системе.* В разделе приводятся требования к структуре АС, численности и квалификации персонала, режимам работы системы. Среди требований могут быть требования к техническому обслуживанию системы, к сохранению и защите информации от несанкционированного доступа и др., а также требования к системе в целом, к функциям системы и к отдельным видам обеспечения.

*5. Состав и содержание работ по созданию системы.* В разделе указывается перечень этапов создания системы, сроки начала и окончания каждого этапа и исполнители работ. В требованиях к составу и содержанию работ перечисляются мероприятия, которые предшествуют внедрению системы, а именно:

- приведение информации к виду, пригодному для обработки на ЭВМ;
- создание необходимых для функционирования информационной системы подразделов;
- срок и порядок комплектования и обучения персонала.

*6. Порядок контроля и приемки системы.* Описываются правила контроля и приемки созданной АС.

*7. Требования к документации системы.* Раздел содержит согласованный с заказчиком перечень документов, которые должны быть разработаны и включать систему учетно-статистической, учетной, финансовой и другой документации. Эта документация содержит следующие виды информации:

*Информация, которая используется*, т. е. приводится перечень массивов информации, которые формируют входные сообщения и сохраняются для реализации данного алгоритма. Для каждого массива приводится его название, идентификатор и возможное количество записей.

*Результативная информация* — описание назначения результатов, перечень массивов информации, которые участвуют в формировании выдаваемых сообщений и сохранении ее для решения других задач.

*Математическое описание* основных показателей задачи, описание процесса и объектов, перечень предварительных оценок разработанной модели при разных условиях работы системы.

*Алгоритм решения* подается в виде схемы в соответствии с требованиями ГОСТ 19.701–90, дополненной текстом в соответствии с ГОСТ 24.301–80 «Система технической документации на АСУ. Общие требования к выполнению текстовых документов».

*Информационное обеспечение* содержит описание: характеристик, организации сбора и передачи информации на обработку, системы классификации и кодирования; структуры документов и информационных массивов.

*Организационное обеспечение* содержит схему технологического процесса автоматизированного сбора, обработки информации, передачи данных с перечнем соответствующей документации.

*Программное обеспечение* включает его характеристику, схему взаимодействия программ и схемы самих программ.

В состав документов рабочего проекта входят такие документы:

- описание программ решения задачи;
- инструкции по технологическому процессу или руководству пользователя;
- классификаторы технико-экономической информации.

*8. Источники разработки.* В разделе перечисляются документы и информационные материалы, которые использовались во время разработки ТЗ, а также те, которые потребуются во время создания информационной подсистемы.

# Приложение Б

# ЖИЗНЕННЫЙ ЦИКЛ КОМПОНЕНТНОЙ РАЗРАБОТКИ ПС

Существует много различных методологий компонентной разработки ПС, которые отличаются методами интеграции, способами описания компонентов, языками определения интерфейсов, способами построения интегрированной среды и др. К основным этапам компонентной разработки ПС относятся:

1) разработка требований (Requirements) к ПС согласно с компонентной методологией;
2) анализ поведения (Behavioral Analysis) ПС для определения сервисных аспектов программы и соответствующие процессы обработки данных;
3) спецификация интерфейсов и взаимодействия компонентов (Interface and Interaction Specification);
4) интеграция компонентов в единую среду для ПС на основе новых компонентов и компонентов повторного применения (Application Assembly and Component Reuse);
5) развертывание (System Deployment) ПС у пользователя;
6) поддержка и сопровождение (System Support and Maintenance) ПС.

Под интегрированной средой понимается результат завершения работы на этапе интеграции, т. е. макета ПС, которая будет функционировать у пользователя. Программная система — это результат работы после этапа развертывания, характеризуется привязкой к компьютерной и сетевой среде пользователя. Кроме этого, компоненты системы могут приобретаться пользователем в отдельности и выполняться самостоятельно.

# Б.1 Этап разработки требований

Разработка требований — это формирование и описание функциональных, технологических, организационных и нефункциональных свойств ПС. Суть этапа состоит в определении бизнес-процессов, выполнение которых должна обеспечить ПС. Бизнес-процессы связаны с профессиональной деятельностью конечного пользователя и их выполнение поддерживается функционированием ПС. Согласно объектно-ориентированному подходу на этом этапе выполняются следующие процессы:

- определение бизнес-процессов;
- формирование прецедентов;
- спецификация требований к бизнес-процессам, условий и особенностей их выполнения.

*Определение бизнес-процессов*. Это наименее формализованный процесс на этапе разработки требований. Начальное описание делается заказчиком ПС с использованием обычного языка представления понятий относительно бизнес-процессов и ПрО. На практике существуют несколько итераций в формировании этого описания, чтобы заказчик и разработчик пришли к соглашению и поняли друг друга.

Метод декомпозиции предусматривает постепенную и планомерную детализацию функционального, пользовательского и технологического аспектов и т. п. Детализация проводится несколькими итерациями и может закончиться на следующем этапе процесса.

*Формирование прецедентов*. Прецедент описывает последовательность событий в профессиональной деятельности пользователя, где применяется ПС. Прецеденты описывают варианты использования системы. Совокупность всех прецедентов определяет общую картину применения ПС для решения задач определенной ПрО. Поскольку они формируются различными категориями пользователей, могут существовать определенные противоречия в их описаниях. Для их предотвращения, при анализе таких прецедентов, им даются определенные приоритеты, согласно которым они ранжируются и структурируются. Важный момент в формировании прецедентов — согласование с описанием бизнес-процессов.

*Спецификация требований к бизнес-процессам.* Спецификации требований не являются конечными, они выполняются на стадии анализа поведения ПС. Далее проводится переход от общих системных требований к требованиям относительно отдельных компонентов. Детализация системных требований к уровню компонентов является основой их спецификации.

# Б.2 Этап анализа поведения ПС

Основная задача этого этапа — определить, что система непосредственно будет делать. Если предыдущий этап описывает бизнес-процессы и специфицирует разные их аспекты, то на этом этапе анализируются детали этих процессов, определяются подходы и методы их поддержки в системе. Данный этап включает такие процессы:

- моделирование предметной области;
- построение объектной системы;
- адаптация объектной системы.

*Моделирование предметной области* определяется степенью формализации и уровнем детализации в предоставлении знаний относительно предметной области. Главные задачи этого процесса являются типичными и состоят:

- в определении базовых понятий и терминов предметной области, а также реальных объектов, которые им отвечают;
- в определении отношений и связей между этими понятиями соответственно отношениям и связям между реальными объектами, которые важны для функциональности будущей ПС;
- в построении моделей, которые отображают различные аспекты предметной области: структурные, поведенческие, последовательности действий, потоки данных и пр.

Результаты моделирования могут уточняться и детализироваться на иных этапах с целью наиболее полного описания предметной области. В частности, важным моментом является согласование терминов и понятий по результатам моделирования и терминологии описания бизнес-процессов.

*Построение объектной системы* состоит в определении структурных свойств для совокупности объектов, ассоциаций между ними, описании их атрибутов и соответствующих значений. Далее определяются характеристики поведения для объектной системы — состояние системы, переходы между состояниями, граф переходов, эволюция объектов и системы в целом.

*Адаптация объектной системы.* Этот процесс не типичный, его сущность состоит в оптимизации объектной системы к возможностям реализации общих функций за счет функций существующих программных компонентов. То есть система трансформируется таким образом, чтобы объекты, которые присутствуют в ней со своими сервисными возможностями, были приближены к функциональным возможностям компонентов.

## Б.3 Этап спецификации интерфейсов и взаимодействия компонентов

Этот этап является ключевым моментом, когда специфика компонентного подхода начинает играть главную роль в создании ПС. Предыдущие этапы повторяют соответствующие этапы объектно-ориентированного подхода. Основу компонентного подхода составляют иные аспекты. Для объектов существуют много реализованных и готовых к применению компонентов, и их необходимо предоставить так, чтобы можно было задействовать механизмы их поиска, выборки и интеграции в единую среду. Эта цель и определяет главную задачу этапа спецификации интерфейсов и взаимодействия компонентов, что соответствует следующим процессам:

- распределению ролей компонентов;
- проектированию и спецификации отдельных интерфейсов;

- описанию взаимодействий компонентов.

*Распределение ролей компонентов.* Интерфейсы специфицируются исходя из ролей и физических реализаций соответствующих объектов системы. Распределение ролей определяет поиск и выбор компонентов, которые имеют соответствующие сервисные возможности и необходимые функции.

*Проектирование и спецификация интерфейсов*. Проектирование интерфейсов происходит соответственно ролям компонентов. Важно придерживаться концепции оптимальности в проектировании — интерфейсов для компонента не должно быть много, но в то же время не нужно проектировать мало, но больших по размеру, интерфейсов. Каждый из типов интерфейсов — клиентский или сервисный — проектируется в отдельности, идентифицируется и определяется состав поддерживаемых ими операций. Описание отдельных интерфейсов проводится в языке IDL для модели CORBA.

Каждый интерфейс составляется из определенного количества операций с типами параметров и результатов, представленных в терминах пред- и постусловий и определения возможных нестандартных результатов их выполнения с помощью специального объекта, который содержит данные о нестандартных ситуациях.

*Описание взаимодействия компонентов* выполняется в контексте последовательности действий (workflow), поддерживающих определенные бизнес-процессы. Если результат проектирования интерфейсов определяет пары взаимодействующих компонентов, то результатом этого процесса является совокупность последовательностей операций всех компонентов для достижения целей выполнения бизнес-процессов.

## Б.4 Этап интеграции

Главными задачами этого этапа являются определение, поиск и выбор всех необходимых компонентов, их адаптация, определение плана компонентной конфигурации, создание и тестирование интегрированной среды для ПС. Поиск и выбор компонентов объединяются в одну задачу, что называется квалификацией (Component Qualification), а интеграция компонентов определяется как композиция, которая разделяется на несколько видов в зависимости от комбинаций компонентов. Входными данными процесса являются описания:

- интерфейсов компонентов, на соответствующем языке описания;
- взаимодействия компонентов последовательности операций для выполнения функций в ПС;
- дополнительных условий и данных для интеграции и управления компонентами.

На этапе интеграции соответственно выполняются следующие процессы:

- поиск компонентов соответственно описанию интерфейсов;
- выбор совокупности компонентов, которые обеспечивают необходимую функциональность;
- адаптация существующих компонентов к требованиям построения интегрированной среды;

- создание новых компонентов, для которых результаты поиска, выбора и адаптации не дали требуемых результатов;
- инсталляция компонентов для потребностей интеграции;
- определение полной совокупности правил и условий интеграции;
- непосредственное построение проекта;
- тестирование интегрированной среды.

*Поиск компонентов*. Этот процесс предусматривает существование информационных хранилищ с описаниями компонентов. Для реализации более качественного и оптимального поиска целесообразно иметь систему классификации программных компонентов.

Для предоставления и поиска информации могут быть применены поисковые машины сети Интернет, как, например, AltaVista, проект Agora, нацеленный на разработку поисковой машины, для которой критериями поиска являются компонентные модели и их свойства.

*Выбор компонентов.* Результаты поиска могут быть неопределенными, т. е. могут существовать несколько компонентов для определенного интерфейса или некоторый компонент почти не подходит. Для таких случаев необходимо выполнить процедуры оценки и выборки, например основанные на системах показателей качества.

*Адаптация компонентов*. Для расширения возможностей компонентов с одновременным сохранением их основных функциональных и технологических характеристик и обеспечения полной реализации интерфейсов, условий взаимодействия и особенностей интегрированной среды проводится адаптация. Основой механизма реализации такой трансформации является свойство к усовершенствованию и расширению своей функциональности и иных возможностей. К корректным механизмам относятся:

- расширение существующих интерфейсов компонентов с целью соответствия интерфейсам в интегрированной среде;
- обеспечение дополнительных реализаций компонентов с сохранением своих интерфейсов;
- создание контейнеров, которые вмещают иные компоненты с одновременным предоставлением их как самостоятельных компонентов интегрированной среды с необходимой функциональностью и интерфейсами.

*Создание компонентов.* Этот процесс выполняется, когда предыдущие процессы поиска, выбора, адаптации для определенных компонентов дают отрицательные результаты. В этом случае необходимо создавать новые компоненты с заранее определенными интерфейсами и функциональностью. Для правильного построения новых элементов используется модель Enterprise java beans (EJB), в состав которой входят:

- сервер EJB, как прикладной сервер, в среде которого загружаются и выполняются один или больше контейнеров EJB;
- контейнер EJB, который отвечает за создание среды и условия функционирования соответствующего компонента;

- специальный интерфейс, который поддерживает выполнение условий жизненного цикла для экземпляров компонента, в частности их создание;
- специальные интерфейсы, которые обеспечивают дополнительные сервисы функционирования компонентов в условиях распределенной обработки для обеспечения целостности выполнения функций, для которых задействованы операции в нескольких компонентах.

*Инсталляция компонентов для потребностей интеграции.* Этот процесс не отличается от инсталляции отдельных компонентов этапа развертывания. Единственное отличие состоит в том, что определенные компоненты специально созданы самым разработчиком ПС для ее потребностей, а, следовательно, условия и процедуры инсталляции он может оптимизировать.

*Определение правил и условий интеграции*. Правила и условия интеграции зависят от компонентной модели, практическое применение имеют три:

- COM/DCOM/COM+, реализованная в операционных системах семейства WINDOWS;
- система CORBA [11], которая поддерживается на различных платформах;
- Javabeans и enterprise javabeans [13] на базе JAVA-технологий, которые функционируют на платформах, для которых реализованы виртуальные JAVA-машины.

*Построение проекта*. Главная цель процесса — выполнить все действия относительно подготовки интегрированной среды к функционированию и создание плана конфигурации компонентов ПС.

*Тестирование интегрированной среды.* При тестировании решаются две задачи:

- проверка функциональности ПС;
- проверка возможности совместимой работы компонентов системы.

Первая задача традиционная: для создания любой программы или ПС проводится проверка корректности функционального назначения и организации взаимодействия с пользователем.

Вторая задача — специфична для компонентного подхода, она имеет факторы, которые влияют на проверку совместной работы компонентов:

1) Инициирование работы интегрированной среды для ПС.
2) Определение инфраструктуры для создания возможности динамического взаимодействия компонентов во время выполнения и распределения клиент-серверных запросов в системе.
3) Для каждой пары компонентов, которые взаимодействуют, один из них является клиентом, а второй — сервером. Серверы функционируют в режиме ожидания, пока клиенты к ним не обратятся. Механизм поддержки режима функционирования на уровне операционной среды является механизмом сервисов.
4) Схема взаимодействия компонентов в ПС является ориентированным графом. Приведенные факторы и определяют порядок и содержание второй задачи для проведения тестирования:

- проверка процедуры старта ПС;
- проверка функционирования и создания инфраструктуры для поддержки компонентной модели;
- проверка корректности регистрации сервисов и возможности взаимодействия с ними с различных компьютеров сети;
- проверка корректности последовательностей инициирования сервисов.

## Б.5 Этап развертывания компонентов системы

На выполнение задачи этого этапа влияет ориентация ПС на конечного пользователя. В случае, когда ПС создается для конкретного заказчика (часто он и является пользователем), некоторые задачи развертывания решаются на предыдущих этапах. К ним относятся:

- проектирование и интеграция ПС с ориентацией на конкретные компьютерные условия заказчика, расположение, наличие компьютерных сетей, коммуникаций и др.;
- инсталляция отдельных компонентов на определенных компьютерах для целей интеграции;
- создание компонентной конфигурации на этапе интеграции с ориентацией на ее дальнейшее применение на этапе сопровождения.

Пользователь компонентной системы должен пройти все процессы этапа развертывания на своей собственной базе соответственно свой конфигурации компьютерных и общесистемных средств.

В состав этапа развертывания входят:

- оптимизация плана компонентной конфигурации соответственно имеющейся в компьютерной базе пользователя, ее свойствам и топологии;
- развертывание отдельных программных компонентов;
- создание целевой компонентной конфигурации.

*Оптимизация плана компонентной конфигурации*. План компонентной конфигурации является результатом тапа интеграции, он определяет абстрактную модель предоставления интегрированной среды ПС. Все взаимосвязи и взаимодействия компонентов представлены на логическом уровне, описываются сами факты связей и взаимодействий без учета реального расположения компонентов.

Главная задача процесса — определения реальной конфигурации компонентов и оптимизационная задача с данными:

- компьютерные мощности и способы коммуникации у пользователя;
- требования к ресурсам со стороны отдельных компонентов;
- план компонентной конфигурации.

В качестве предельных условий выступает необходимость выполнения функциональных, технологических и иных требований к программной системе.

*Развертывание программных компонентов* путем инсталляции компонентов на конкретных компьютерах, расположение которых определено в результате предыдущего процесса. Процесс инсталляции для каждого отдельного компонента может иметь свои собственные отличия, которые зависят от метода и способов инсталляции.

*Создание целевой компонентной конфигурации.* Результаты предыдущих процессов являются основой для создания целевой конфигурации, в задачи которой входит:

- расположение отдельных компонентов (сетевые адреса, имена каталогов и файлов);
- характеристики компонентов (например, размеры, необходимые ресурсы и условия функционирования и др.);
- описание взаимосвязей компонентов (например, описание последовательностей инициирования и окончания работы).

Этот процесс завершает создание целевой конфигурации. Предыдущая информация дополняется:

- организационными требованиями (приведение в порядок категорий пользователей, предоставление им определенных прав доступа к отдельным компонентам, к сервисам и др.);
- технологическим регламентом (например, по времени и периодичности инициирования компонентов, выполнению определенных задач и др.);
- описанием процесса инициирования ПС в целом (например, может быть специально сформированный пакет для запуска и выполнения системы).

## Б.6 Этап сопровождения

В сравнении с традиционными методологиями разработки ПС этап сопровождения в компонентной методологии характеризуется следующими особенностями.

1) Обслуживающий персонал ПС не имеет доступа к коду компонентов. В связи с этим при необходимости изменения ПС традиционные подходы и методы адаптируются к возможностям новых условий функционирования, тестирования, выявления и исправления ошибок, модификации отдельных элементов и др.
2) Вся политика модернизации, усовершенствования, расширения ПС должна строиться на компонентной основе, в которой главными механизмами могут лишь быть:
   - замена существующих компонентов новыми компонентами с сохранением интерфейсов и сервисных возможностей;
   - расширение функциональных и технологических возможностей отдельных компонентов на основе их свойств и сохранение существующих интерфейсов.
3) Отдельные компоненты, которые применяются в ПС, могут быть созданы посторонними разработчиками и использоваться в данной ПС, как готовые.

Соответственно с этими производителями проводится собственная политика относительно поддержки, усовершенствования, развития таких компонентов. При сопровождении ПС такие ситуации необходимо учитывать как в технологическом, так и в организационно-правовом аспекте (например, охрана авторских прав на программное обеспечение).

Эти особенности существенным образом влияют на традиционные задачи этапа сопровождения и процессов, которые их поддерживают. К основным процессам этого этапа относятся:

- модификация компонентной конфигурации;
- адаптация новых компонентов к требованиям в условиях интегрированной среды;
- анализ отказов функционирования, обнаружение дефектов, поиск и исправление ошибок в программной системе;
- тестирование ПС.

Кратко остановимся на общей характеристике этих процессов.

*Модификация компонентной конфигурации.* Этот процесс отвечает за следующее:

- добавление и исключение определенных компонентов;
- замещение существующих компонентов новыми как с тождественной функциональностью и интерфейсами, так и с расширенными характеристиками.

Необходимыми условиями для этого процесса является возможность манипуляций с компонентами как отдельными объектами с сохранением свойств и характеристик разных частей ПС. Это достигается благодаря применению систем управления конфигурациями, с помощью которых отслеживаются и выполняются все изменения в конфигурации системы.

*Адаптация новых компонентов к требованиям и условиям среды*. Данный процесс, по сути и по содержанию, почти не отличается от соответствующего процесса этапа интеграции. Имеющиеся отличия носят непринципиальный характер. К ним, в частности, можно отнести то, что в случае неудовлетворительной адаптации всегда имеется возможность вернуться к существующему компоненту и программная система остается без изменений.

Кроме этого, сам процесс адаптации может выполняться обслуживающим персоналом пользователя (при наличии специалистов с необходимой квалификацией), а не разработчиком ПС.

*Анализ отказов функционирования, поиск и исправление ошибок в ПС*. Если при определенных условиях в программной системе появляются отказы функционирования или ошибки программирования, то главной задачей их локализации является нахождение компонентов, которые ненормально работают. В большинстве случаев обслуживающий персонал не в состоянии исправить код компонента, к которому нет доступа.

Для исправления ошибок используются следующие механизмы:

- обращение к разработчику компонента и, если компонент был специально создан для этой системы, ожидание от разработчика исправления ошибки, а потом замена соответствующего компонента.

- если компонент является коммерческим продуктом, который создан сторонним производителем, то соответствующие разработчики должны сообщить об этом и дождаться официальной версии компонента, в котором исправлена ошибка, а затем заменить этот компонент;
- не дожидаясь исправления ошибки другими разработчиками, самостоятельно провести замену локализованного ошибочного компонента другим правильным с соответствующей функциональностью и интерфейсами.

На период исправления ошибки последовательности взаимодействий компонентов, которые были определены на предыдущих этапах, выключаются из функционирования путем внесения адекватных изменений в компонентную конфигурацию.

*Тестирование ПС*. Тестирование проводится периодически для проверки правильности функционирования системы и в случаях внесения изменений в компонентную конфигурацию или замены отдельных компонентов. Под этим процессом понимается тестирование системы на компонентном уровне, т. е. покомпонентное тестирование.

Для проведения периодического тестирования применяются тесты, которые передаются разработчиком ПС пользователю. Главная цель такого тестирования — подтвердить правильность функционирования системы, оценить ее эффективность, быстродействие и другие технологические характеристики. Порядок и условия процесса тестирования для ПС отображаются в соответствующих документах.

Для тестирования проведенных изменений в компонентную конфигурацию и отдельные компоненты необходимо иметь:

- сами компоненты в готовом к применению виде;
- гарантию относительно достаточно полного тестирования компонентов их разработчиками, информацию о результатах тестирования, в частности перечень еще неисправленных ошибок;
- четко сформулированные условия применения компонентов как с функциональной точки зрения, так и с технологической (в частности, иметь данные о ресурсах, необходимых для нормальной работы компонентов системы).

При разработке тестов учитываются:

- последовательности взаимодействий компонентов, в состав которых входят те компоненты, которые проходят тестирование;
- информация о самостоятельном тестировании компонентов (используется для уменьшения объема тестирования);
- условия для нормального функционирования компонентов.

После проведения тестирования и анализа результатов, при наличии ошибок разного рода, вносятся соответствующие изменения как в компонентную конфигурацию, так и в отдельные компоненты системы, в которых обнаружены ошибки.

# Приложение В

# КОДЕКС ЭТИКИ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Своим появлением программная инженерия обязана деятельности мощных профессиональных объединений — The Association for Computer Machinery (ACM) и Institute of Electrical and Electronics Engineers Computer Society (IEEE Computer Society). Общими усилиями этих двух объединений разработан кодекс этики программной инженерии, который фокусирует мораль, правила и нормы поведения профессионалов, их обязательства и ответственность по отношению к обществу и друг к другу.

Этика инженерной деятельности в программировании отличается от этики прикладных исследований, где исследователи работают в прикладной науке, направляют свои усилия на реализацию возможностей и отвечают определенным требованиям.

Инженерная деятельность в программной инженерии, кроме сказанного, включает технические умения, ответственность перед пользователями, умение управлять и приводить к удачному завершению больших программных проектов. Иначе говоря, инженеры должны хорошо знать, что есть выбор сделать реализацию быстро, с высокой степенью риска, либо высококачественно, с малым риском.

Кодекс состоит из преамбулы и восьми принципов, которых должны придерживаться профессионалы программной инженерии. В преамбуле профессионал определяется как специалист, который принимает непосредственно участие в деятельности или в обучении анализу, спецификации, проектированию, разработке, сертификации, сопровождению и тестированию программных систем.

Сформулированные принципы декларируют здоровье, безопасность и благосостояние общества как главный фактор, который необходимо принимать во внимание при принятии решений в профессиональной деятельности программной инженерии.

В кодексе задекларировано восемь принципов, которые касаются соответственно:

1) согласования профессиональной деятельности с интересами общества;

2) взаимоотношений между клиентом, работодателем и исполнителем разработки;
3) достижения соответствия качества продукта лучшим профессиональным стандартам;
4) соблюдения честности и независимости при профессиональных оценках;
5) соблюдения этических норм в менеджменте и в сопровождении разработок;
6) поддержки становления профессии в соответствия с кодексом этики;
7) соблюдения этических норм во взаимоотношениях между коллегами;
8) усовершенствования специальности.

Каждый из приведенных принципов имеет детальные пояснения относительно различных спектров его соблюдения.

# Приложение Г

# СТАНДАРТЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Существует определенное количество организаций, профилирующими направлениями деятельности которых являются разработка и сопровождение стандартов. В зависимости от области применения стандарт может иметь статус международного, ведомственного или стандарта предприятия.

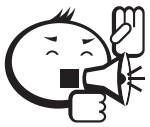
Главным органом установления международных стандартов является международная организация по стандартизации (The International Standards Organization или, сокращенно, ISO), которая работает в сотрудничестве с международной электротехнической комиссией (The International Electrotechnical Commission или, сокращенно, IEC).

Все утвержденные ими совместно стандарты имеют идентификатор, который состоит из префикса ISO/IEC, серийного номера стандарта и даты выпуска, например: ISO/IEC 12207: 1995–08–11.

Каждый стандарт ISO/IEC имеет также название, которое при ссылках указывается после идентификатора. ISO/IEC имеет десятки технических профильных комитетов, в частности технический комитет «Информационные технологии», одним из технических подкомитетов которого является подкомитет по программной инженерии. Существуют также международные объединения по отдельным проблемным областям, которые выпускают стандарты для соответствующих приложений. Каждое цивилизованное государство имеет свои национальные органы стандартизации.

Большинство национальных комитетов по стандартизации признают стандарты ISO/IEC, входят в ее состав и проводят для стандартов ISO/IEC процедуру

гармонизации, т. е. их приспособление к национальным условиям и особенностям применения (как, например, национальные алфавиты, метрические системы, валютные знаки и т. п.).



Главным источником стандартов являются профессиональные объединения, в частности для программной инженерии — IEEE Computer Society.

На данное время существует свыше 300 стандартов IEEE для программной инженерии, значительная часть которых принимается во внимание широким кругом разработчиков программных систем. Практически большинство стандартов программной инженерии исторически появляются как стандарты IEEE, а со временем, после испытания опытом использования, вносятся как кандидаты в стандарты ISO/IEC.

Процедура утверждения стандартов ISO довольно сложная. Имеется несколько стадий прохождения кандидата в стандарт, для любой из них предусмотрена рассылка предложений на экспертизу всем национальным комитетам, сбор замечаний, их обработка и голосование для создания новой версии.

Эта процедура для некоторых стандартов может длиться годами относительно официальных стандартов или стандартов де-юре. Бюрократизированная процедура приводит к тому, что технически доведенные до кондиции стандарты могут за какое-то время утратить свою значимость для индустрии программного обеспечения и соответствие действующему уровню технологии.

Тем временем, индустрия создает так называемые «стандарты де-факто», которые фактически находят массовое использование независимо от того, утверждены они компетентными плановыми органами или нет, так как они являются наиболее актуальными в индустрии программных систем.

Термином «стандарт де-факто» обозначаются спецификации на проект стандарта (или внутренний стандарт), которые публикуются некоторым консорциумом (группой учреждений, которые официально работают для общей цели стандартизации) или фирмой как получившие общее одобрение от других разработчиков, вследствие чего этот проект стал восприниматься как необходимое современное направление технологии.

Например, проектные решения консорциума Object Management Group (OMG) по управлению транзакциями стали стандартом де-факто и утверждены комитетом ISO, модель UML фирмы Rational Rose определена как стандарт моделирования артефактов программной инженерии, язык JAVA — претендент на стандарт, который активно используется на рынке, и др.

Стандарты де-факто, благодаря своему соответствию актуальным потребностям рынка, распространяются и внедряются довольно быстро, потому что их отработка и апробация проходит внутри групп, которые их создают, а публикация этих стандартов обозначает их как «зрелые» решения.

Учитывая вышесказанное, влиятельные международные организации и, в первую очередь, ISO, которые являются разработчиками стандартов де-юре, признали, что они не являются монопольными и компетентными источниками всех стандар-

тов и поэтому ввели процесс преобразования стандартов де-факто в стандарты де-юре, которые в этом случае получили название общедоступной спецификации (Public-Available Specifications — сокращенно PAS).

*Объекты стандартизации.* Согласно эталонной модели программной инженерии в ней определены такие главные элементы :

- процессы разработки ПО;
- продукты разработки;
- ресурсы, которые используют процессы для создания программного продукта.

Важным элементом моделирования проблем предметной области является клиент (заказчик), который заказывает программную систему. Требования заказчика определяют состав и качество указанных элементов. Объектами стандартизации любого стандарта в программной инженерии являются аспекты указанных выше элементов или их соединений.

Больше всего стандартов существует для разных видов процессов. Так, в стандарте ISO/SEC 12207 базовых процессов — 42, а всего процессов в нем более 200. Надо сказать при этом, что усовершенствование процессов, как правило, ведет к усовершенствованию создаваемых продуктов и эффективному использованию ресурсов.

Определен детальный перечень процессов и действий, которые составляют процессы, для всех этапов жизненного цикла разработки и большинства аспектов рассмотрения указанных этапов. Наибольший успех и широкое использование приобрел стандарт [2] для процессов жизненного цикла программного обеспечения. Этот стандарт стал определенным каркасом для рассмотрения всех проблем программной инженерии.

Первым измерением классификации является отношение стандарта к продукту, процессу, ресурсу или взаимодействию с заказчиком. Иногда рассматривают классификацию стандартов по уровням обобщения регламентаций, которые подаются в стандарте.

Отдельные группы разработчиков стандарта в программной инженерии создают:

- терминологию,
- общие рекомендации,
- принципы действий,
- стандартизированные элементы,
- рекомендации для прикладных применений,
- рекомендации относительно инструментов и методов разработки (например, классификация аномалий выполнения),
- планы управления качеством,
- планы валидации и верификации.

В соответствии с предложенной классификацией ниже приведены наиболее значимые и существующие стандарты программной инженерии.

ISO/IEC 12207:1996 Information Technology — Software life-cycle processes;

12207/FPDAM 1.2 — Software Engineering — Life Cycle Processes // ISO/IEC;

JTC1/SC7 N2413, Software &System Engineering Secretariat, Canada. — 2001. — 62 p.;

DTR 15271 Information Technology — Guide for ISO/IEC 12207 (Software Life Cycle Processes);

IEEE/IEA Std. 12207.1:1997 Software Life Cycle processes — Life Cycle data;

ISO/IEC JTC1/SC7 N2172a (Draft ISO/IEC TR 16326:1999). Software Engineering — Guide for the Application of ISO/IEC 12207 to Project Management;

ISO/IEC TR 15504 Information Technology — Software Process Assessment Part 1–Part 9;

IEEE Std. 1058:1998 IEEE Standard for Software Project Management Plans;

IEEE Std. 1044:1993 IEEE Standard Classification for Software Anomalies;

IEEE Std. 1044–1:1995 IEEE Guide to Classification for Software Anomalies;

ISO/IEC 15026:1998 Information Technology — System and Software Integrity Levels;

ISO/IEC JTC1/SC7 N2207:1999 IEC 60300: Dependability management — Part 3–13: Application Guide — Project risk management;

ISO/IEC JTC1/SC7 N2198:1999 2nd Working Draft On Risk Management Terminology и др.

# Приложение Д

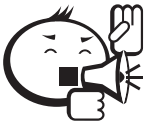
# ЖИЗНЕННЫЙ ЦИКЛ ПО СТАНДАРТА ISO/IEC 12207 И СВЯЗЬ ЕГО С ЯДРОМ ЗНАНИЙ ПРОГРАММНОЙ ИНЖЕНЕРИЕЙ SWEBOK

Программная инженерия как инженерная дисциплина охватывает все аспекты создания ПО от начальной стадии разработки системных требований, создания ПО и его использования. Эталонная модель программной инженерии включает три взаимосвязанных фактора: процессы, программные продукты, ресурсы (человеческие, технические и финансовые).

Каждая ПС на протяжении своего существования проходит определенную последовательность процессов (этапов), начиная от постановки задачи до ее воплощения в готовую программу, эксплуатации и изъятия. Такая последовательность этапов называется жизненным циклом (ЖЦ) разработки ПС. На каждом этапе ЖЦ выполняется определенная совокупность процессов и/или подпроцессов, каждый из которых порождает соответствующий промежуточный продукт, используя результаты предыдущего.

Все продукты процессов программной инженерии представляют собой некоторые описания, а именно: тексты требований к разработке, согласование договоренностей с заказчиком, архитектуру, структуру данных, тексты программ, документацию, инструкции по эксплуатации и т. п.

Главными ресурсами разработки ПС в программной инженерии являются сроки, время и стоимость. Правильное использование этих ресурсов на процессах ЖЦ определяет эффективность этой разработки.



Разновидности действий и задач, представленные в процессах ЖЦ ПС, отображены в международном стандарте ISOIEC 12207 (таблица Д.1) и связаны содержательно с областями знаний SWEBOK.

Данный стандарт устанавливает архитектуру верхнего уровня ЖЦ ПО, начиная от разработки концепции до утилизации системы. Архитектура представляет собой множество процессов, взаимосвязей между ними и определяет действия и задачи, т. е. определяет, что надо делать, а не как надо выполнять действия или задачи процессов.

Стандарт не обязывает использовать определенную модель ЖЦ ПО или конкретную методологию разработки ПО и не предъявляет требования к формату и содержанию создаваемых документов. Поэтому организации-пользователю этого стандарта потребуются для своей работы дополнительные стандарты или процедуры, определяющие разные детали процесса (ISO выпускает руководства и процедуры, дополняющие стандарт 12207).

Основная идея данного стандарта состоит в том, что разработка и сопровождение ПО должны осуществляться так, как этого требует инженерная дисциплина. Следуя этой идее, разрабатывается каркас (framework), имеющий четкие связи с окружением системной инженерии — ПО, техническим обеспечением, исполнителями и деловой практикой.

Все процессы в данном стандарте разделены на три категории:

- основные процессы;
- обеспечивающие (поддерживающие) процессы;
- организационные процессы.

Для каждого из процессов определены виды деятельности (действия — activity) и задачи и определяется совокупность результатов (выходов) видов деятельности и задач, а также некоторые специфические требования. Стандарт дает перечень работ для основных обеспечивающих и организационных процессов.

Таблица Д.1 – Процессы ЖЦ в стандарте ISO/IEC 12207

| **№ п/п** | **Наименование процессов (подпроцессов)** |
|---|---|
| **Категория «Основные процессы»** | |
| 1.1 | Заказ (договор) |
| 1.1.1 | Подготовка заказа, выбор поставщика |
| 1.1.3 | Мониторинг деятельности поставщика, прием потребителем |
| 1.2 | Поставка (приобретение) |
| 1.3 | Разработка |
| 1.3.1 | Выявление требований |
| 1.3.2 | Анализ требований к системе |
| 1.3.3 | Проектирование архитектуры системы |
| 1.3.4 | Анализ требований к ПО системы |
| 1.3.5 | Проектирование ПО |
| 1.3.6 | Конструирование (кодирование) ПО |
| 1.3.7 | Интеграция ПО |
| 1.3.8 | Тестирование ПО |
| 1.3.9 | Системная интеграция |
| 1.3.10 | Системное тестирование |
| продолжение на следующей странице | |

Таблица Д.1 – Процессы ЖЦ в стандарте ISO/IEC 12207

| № п/п | Наименование процессов (подпроцессов) |
|---|---|
| 1.3.11 | Инсталляция ПО |
| 1.4 | Эксплуатация |
| 1.4.1 | Функциональное использование |
| 1.4.2 | Поддержка потребителя |
| 1.5 | Сопровождение |
| **Категория «Процессы поддержки»** | |
| 2.1 | Документирование |
| 2.2 | Управление конфигурацией |
| 2.3 | Обеспечение гарантии качества |
| 2.4 | Верификация |
| 2.5 | Валидация |
| 2.6 | Общий просмотр |
| 2.7 | Аудит |
| 2.8 | Решение проблем |
| 2.9 | Обеспечение применимости продукта |
| 2.10 | Оценивание продукта |
| **Категория «Организационные процессы»** | |
| 3.1 | Категория |
| 3.1.1 | Управление на уровне организации |
| 3.1.2 | Управление проектом |
| 3.1.3 | Управление качеством |
| 3.1.4 | Управление риском |
| 3.1.5 | Организационное обеспечение |
| 3.1.6 | Измерение |
| 3.1.7 | Управления знаниями |
| 3.2 | Усовершенствование |
| 3.2.1 | Внедрение процессов |
| 3.2.2 | Оценивание процессов |
| 3.2.3 | Усовершенствование процессов |

Основные процессы:

- процесс приобретения инициирует ЖЦ ПО и определяет действия организации-покупателя (или заказчика), которая приобретает автоматизированную систему, программный продукт или сервис. Этот процесс включает следующие виды деятельности: инициацию; подготовку запроса, контракта и его актуализацию; мониторинг поставщиков; приемку и завершение;
- процесс поставки определяет действия предприятия-поставщика, которое снабжает покупателя системой, программным продуктом или сервисом. Данный процесс включает в себя следующие виды деятельности: инициацию; подготовку предложений (ответа на запрос); контракт; планирование; выполнение и контроль; анализ и оценку; поставку и завершение. Процесс поставки начинается тогда, когда устанавливаются договорные отношения на поставку ПО между заказчиком и поставщиком. В зависимости от усло-

вий договора процесс поставки может включать процесс разработки ПО, процесс эксплуатации для обеспечения служб эксплуатации ПО или процесс сопровождения для исправления и улучшения ПО;

- процесс разработки определяет действия предприятия-разработчика, которое разрабатывает программный продукт. Этот процесс включает в себя: внедрение процесса (implementation); анализ требований к системе; проектирование архитектуры системы; анализ требований к ПО; проектирование архитектуры ПО; детальное проектирование ПО; кодирование и тестирование ПО; интеграцию ПО; интеграцию системы; квалификационное тестирование; установку ПО; обеспечение приемки ПО;
- процесс эксплуатации определяет действия предприятия-оператора, которое обеспечивает обслуживание системы (ПО) в процессе ее эксплуатации пользователями (консультирование пользователей, изучение их потребностей с точки зрения удовлетворения их системой и т. д.). Этот процесс направлен на: внедрение процесса; функциональное тестирование; эксплуатацию системы; обеспечение пользователя документацией по проведению эксплуатации ПО;
- процесс сопровождения определяет действия организации, выполняющей сопровождение программного продукта (управление модификациями, поддержку текущего состояния и функциональной пригодности, инсталляцию и удаление программного продукта на вычислительной системе пользователя). Данный процесс ориентирован на: внедрение процесса; анализ проблем и модификацию; реализацию модификаций; анализ сопровождения; миграцию (перемещение) ПО; удаление ПО.

К обеспечивающим процессам создания ПС относятся: документирование, управление версиями, верификация и валидация, просмотры, аудиты, оценивание продукта и др. Процессы управления версиями соответствуют управлению конфигурацией системы, которая, так же как и продукты, процессы, должна проверяться на правильность реализации целей проекта и соответствия требованиям заказчика. Задачи проверки рекомендуется выполнять специальным контролерам, обладающим знаниями методов и процессов.

К организационным процессам относятся процессы управления проектом (менеджмент разработки), качеством, риском и др. Эти процессы организационно поддерживаются специальными службами: контроля процессов, измерения продуктов, проверки качества, соблюдения стандартных положений и др. Предполагается проведение обучения персонала, определение набора задач и ответственности каждого участника в реализации задач на процессах ЖЦ и др.

Процессы, определенные в этом стандарте, образуют полное множество. Пользователь стандарта может выбрать соответствующее подмножество для достижения своей конкретной цели. Процессы, действия и задачи приведены в стандарте в наиболее общей естественной последовательности. В зависимости от целей конкретного проекта процессы, действия и задачи выбираются, упорядочиваются и применяются итерационно или рекурсивно. Разработчик должен определить или выбрать модель ЖЦ ПО в зависимости от сложности, стоимости и ресурсов программного проекта.

Данный стандарт является главным документом, определяющим содержание деятельности в сфере технологии разработки, а знания, которые необходимы исполнителям для выполнения всех видов деятельности по проектированию и реализации поставленных задач перед проектом, определяют методы и средства ядра знаний SWEBOK.

Между стандартом ISOIEC 12207 и ядром знаний SWEBOK существует связь и взаимовлияние друг на друга, тем более в разработке обоих документов примерно в одно время принимали участие высококвалифицированные специалисты в области программирования и информатики.

Общие идеи и методы программирования, сложившиеся в 90-х годах прошлого столетия, проникли в оба направления и оказали влияние на их структуру и содержание. Программисты-профессионалы систематизировали накопившиеся знания и создали 10 разделов, которые близки процессам ЖЦ по целям, задачам и видам деятельности. В ядре знаний SWEBOK они изложены как фундаментальные знания и инженерные методы управления разработкой ПО, а в стандарте — как общие положения, структура и регламентированные процессы проектирования, начиная от процесса постановки требований до эксплуатации ПО. Процессы стандарта отвечают на вопрос, как надо делать, т. е. какие действия и задачи процессов ЖЦ надо выбрать, чтобы построить конкретное ПО. Ядро знаний SWEBOK отвечает на вопрос, какими методами, средствами и инструментами надо выполнять регламентированные действия и задачи процессов ЖЦ, чтобы построить ПО.

Таким образом, программная инженерия сформировалась как инженерная дисциплина, которая базируются на теоретических и прикладных методах и средствах разработки ПО, которые будут излагаться более подробно в данном учебнике и стандартах (ISO/IEC 12207, 15404, ISO 9126 и др.), содержащих рекомендации, правила и методики управления разработкой ПО. Эти два базиса объединяет инженерия оценивания результатов на процессах ЖЦ, управление качеством ПО, оценка затраченных ресурсов на его создание и учет стоимости деятельности участников разработки.

Таким образом, инженерия программирования делает акцент на принципы, методы и подходы к управлению проектом, конфигурацией и качеством ПО, а стандарты регламентирует процессы организационной деятельности по инженерному проведению работ в процессе проектирования и разработки ПО.

Ядро знаний SWEBOK, а также многочисленные монографии и статьи по методам и средствам программной инженерии предоставляют всю необходимую информацию для выбора наиболее подходящего метода, средства, инструмента, а также процессов ЖЦ для реализации конкретного программного проекта на инженерной и регламентированной, стандартизованной основе.

Естественно, что в небольших программных проектах всегда будет место творческим и неформальным подходам, вносимым отдельными личностями-профессионалами, при создании разного рода уникальных продуктов, процесс разработки которых не всегда укладывается в общее стандартное русло.

Инженерная дисциплина проектирования включает организационные процессы — планирование, управление и сопровождение. Планирование ставит своей целью составить планы и графики работ по реализации проекта и распределить их между разными категориями специалистов с учетом их квалификации и уровня

знаний проблематики программной инженерии. Второй процесс обеспечивает привнесение методов управления в процесс выполнения работ по программированию, а именно управление временем, стоимостью и сроками. Третий процесс рассматривается как процесс выполнения проекта, обнаружения и устранения найденных недостатков в изготовленной системе, а также внесения новых функций по заказу пользователей этой системы. Один из мэтров программной инженерии М. Джексон определил золотое правило программирования: всякая только что законченная программная система сразу требует изменений.

# ГЛОССАРИЙ

До сих пор нет единого международного терминологического словаря, позволяющего работать в едином понятийном пространстве, а ряд терминов не имеет аналогов в национальных языках. Далее приводим наиболее часто встречающиеся слова во многих книгах и Интернет изданиях.

*Абстракция* — способность отделить существенные черты предмета (объекта) от второстепенных, видеть идею, которая будет реализована.

*Абстрактная архитектура* — декомпозированная структура задач домена на подсистемы или иерархию подсистем, на каждом уровне которой фиксируются параметры и ограничения, определяющие выделенные подсистемы.

*Агрегация* — объединение ряда понятий в новое понятие (отношение типа «часть-целое»), общие признаки которого могут быть суммой признаков других компонентов или быть новым признаком.

*Акторы* — действующие лица, которые управляют работой системы.

*Анализ требований* — отображения функций системы и ее ограничений в модели предметной области.

*Артефакт* — любой продукт деятельности специалистов по разработке ПО.

*Архитектура программной системы* — структура системы в терминах подсистем (компонентов) и интерфейсов между ними, отображающая правила декомпозиции проблемы.

*Ассоциация* — наиболее общее и существенное отношение, которое устанавливает наличие связей между понятиями без уточнения их содержания и размеров.

*Белого ящика метод* — исследование внутренней структуры системы в целях выявления ошибок на всех указанных ее путях и потоках передач управления.

*Валидация* — проверка соответствия разработки программной системы требованиям заказчика.

*Верификация* — проверка правильности реализации функций системы на каждом этапе жизненного цикла с учетом требований.

*Взаимодействие объектов* — связь между объектами через механизмы посылки сообщений друг другу.

*Водопадная (каскадная) модель* — схема работ, в которой каждая из работ выполняется один раз и в том порядке, который указан в модели жизненного цикла.

*Гарантия качества программного обеспечения* — действия на каждом этапе жизненного цикла по проверке и подтверждению достигаемого качества соответственно стандартам и процедурам.

*Дефект* — это ошибочное событие в работе системы, возникшее в результате неверного описания спецификации требований, проектных спецификаций, документации и т. п.

*Диаграмма* — графическое представление сценариев работы системы с помощью классов, состояний, событий и т. п.

*Динамическое тестирование* — выполнение программы для обнаружения ошибок, определения их причины и устранения.

*Домен предметной области* — спектр задач проблемы, которые допускают похожие приемы их решения.

*Жизненный цикл системы* — схема выполнения работ по проектированию системы, начиная с момента принятия решения о необходимости ее создания и заканчивая моментом ее полного изъятия из эксплуатации.

*Задача системы* — способ (технология) достижения цели системы с конкретными числовыми (в том числе временными) характеристиками.

*Имитационное моделирование* — моделирование поведения системы в различных аспектах и в разных внешних и внутренних условиях анализа динамических характеристик и распределения ресурсов.

*Инженерия* — применение научных результатов и дисциплины управления программированием задач в целях получения пользы от свойств продуктов, способов взаимосвязи и выполнения.

*Инженерия качества* — процесс управления предоставлением продуктам программного обеспечения свойств качества (надежность, отказоустойчивость и т. п.).

*Инженерия требований* — сбор, анализ, оформление условий и ограничений на разработку системы в виде спецификации, согласованной как заказчиком, так и исполнителем.

*Интенсивность отказов* — это частота появления отказов или дефектов в программной системе при ее тестировании или эксплуатации.

*Инспекция кода* — формальная проверка описания, используемых типов и структур данных в проекте системы на их соответствие требованиям.

*Информационная модель* — модель системы, в которой отображается структура данных и связей с объектами, которые ей пользуются.

*Информационная система* — система, которая выполняет сбор, обработку, сохранение и производство информации с использованием автоматизированных процессоров и людей.

*Информационное обеспечение* — набор средств для предоставления информации пользователям о содержании и условиях ее применения.

*Интерфейсный объект* — стыковочная программа, содержащая описание передаваемых данных, операторы передачи параметров через сообщения для выполнения другого объекта и передачи обратно полученных результатов.

*Инцидент* — абстрактное событие, влияющее на изменение состояния объекта.

*Каркас (фрейм)* — разновидность абстрактной архитектуры для определения выделенных компонентов.

*Качество программного обеспечения* — совокупность свойств, которые определяют пригодность программного обеспечения удовлетворить требования заказчика.

*Компонент* — тип, класс, проектное решение, документация или иной продукт программной инженерии, приспособленный для практического использования.

*Компонентная разработка* — конструирование программного обеспечения путем композиции готовых компонентов, сохраняемых в каталогах.

*Конкретизация* — добавление существенных признаков для расширения содержания некоторого понятия и сужения объема понятия.

*Конечные пользователи системы* — профессиональные лица, которые заказывают компьютерную систему и пользуются ею.

*Конфигурация* — вариант (версия) изготовленной программной системы из отдельных экземпляров компонентов и подсистем.

*Концептуальное моделирование* — процесс построения модели проблемы, ориентированной на понимание ее человеком.

*Критерий* — количественная или качественная характеристика системы, позволяющая оценить степень достижения цели и сформулировать правила выбора необходимых средств (способов, технологий).

*Критерий эффективности* — критерий, позволяющий оценить степень достижения цели с учетом произведенных затрат различных ресурсов.

*Менеджмент* — профессиональное управление коллективами работников (персоналом) при разработке программного проекта.

*Метрика* — количественная мера и шкалы измерения характеристик программы.

*Модель ЖЦ* — типовая схема последовательности работ на процессах разработки некоторого типа программного продукта.

*Модель процесса* — определенная последовательность действий, сопровождающая изменение состояния программного объекта.

*Модель состояний* — отображение динамики изменения состояния объекта класса, которое изменяет его поведение.

*Модель качества* — четырехуровневая структура, которая отображает характеристики, атрибуты, метрики и оценочные элементы характеристик программного обеспечения.

*Надежность программной системы* — это способность системы сохранять свои свойства (безотказность, устойчивость и др.) в процессе преобразования исходных данных в результаты в течение определенного промежутка времени при определенных условиях эксплуатации.

*Нефункциональные требования* — требования, которое характеризуют организационные, исполнительские, операционные аспекты работы программной системы в среде реализации.

*Наследование* — конкретизация в подклассе отдельных свойств, которыми могут пользоваться другие объекты суперкласса.

*Отладка* — проверка программы на наличие в ее описании ошибок и их устранение без внесения новых.

*Отказ* — переход программы из работающего состояния в неработающее состояние в связи с обнаруженными ошибками или дефектами в ней.

*Обобщение* — сужение истинных признаков понятия для расширения свойств, охваченных этим понятием объектов.

*Ошибка* — недостатки в операторах программы или в технологическом процессе ее разработки, которые приводят к неправильной интерпретации исходной информации и к неверному решению.

*Объекты управления* — это функции преобразования объектов интерфейса в объекты сущности аналогично отображению алгоритма обработки данных в системе.

*Объект сущность* — долго живущие объекты, которые отвечают реальным предметам мира предметной области и сохраняют свое состояние после выполнения работы согласно сценарию.

*Объектно-ориентированная модель* — структура из совокупности объектов, которые взаимодействуют между собой, обладают свойствами и поведением.

*Онтология* — совокупность элементарных понятий, терминологии и парадигмы их интерпретации в среде проблемы, которую требуется разработать.

*Оценочный элемент метрики* — количественная или качественная мера оценки соответствующего показателя с учетом его веса в системе оценки качества.

*Оценивание качества* — действия, направленные на определение степени удовлетворения программного обеспечения требованиям, соответствующим его предназначению.

*Пакет* — программная структура с общим механизмом организации элементов (объектов, классов) в группы, начиная от системы (стереотип «система») и к ее подсистемам различного уровня детализации.

*Переносимость системы* — возможность изменять сервис системы (ОС, связи, сетевые коммуникации, данные СУБД и т. п.) путем настройки модулей на новые условия среды или платформы.

*План тестирования* — описание стратегии, ресурсов и графика тестирования отдельных компонентов и системы в целом.

*Поведение домена* — переход элементов домена из состояния к состоянию во времени.

*Повторное использование* — использование в качестве готовой порции любых формализованных знаний, полученных при реализации программных систем.

*Повторно используемый компонент (ПИК)* — фрагмент знаний о минувшем опыте программирования системы, представленный так, чтобы его можно было использовать не только его разработчиками, но и пользователями после соответствующей адаптации к новой среде.

*Прикладная система* — продукт программной инженерии, предназначенный для выполнения конкретных задач конечного пользователя.

*Прецедент (класс)* — определяет набор экземпляров класса, который представляет последовательность действий, выполняемых системой, и выдает результат, ценный для конкретного субъекта.

*Прецедент (экземпляр)* — последовательность действий, выполняемых системой, которая выдает результат, ценный для конкретного субъекта.

*Принципы* — базовые концепции, лежащие в основе всей области программирования.

*Приложение* — области применения, в которых принципы, методы и практика находят свое наилучшее выражение.

*Программная инженерия* — система методов, средств и дисциплины планирования, разработки, эксплуатации и сопровождения программного обеспечения, способного к массовому воспроизводству.

*Процесс приобретения* — действия, которые инициируют определенный цикл анализа для определения покупателем программной системы или сервиса.

*Процесс разработки* — действия разработчика по инженерии требований к проектированию, кодированию и тестированию программного продукта.

*Процесс сдачи* — действия по передаче разработанного продукта покупателю.

*Процесс эксплуатации* — действия по обслуживанию системы пользователем.

*Процесс сопровождения* — действия по решению задач системы, поддержки системы в актуальном состоянии для выполнения функций системы, по управлению модификациями или изъятию системы из употребления.

*Проектирование* — преобразования требований в последовательность проектных решений и в архитектуру системы.

*Проектирование концептуальное* — уточнение понимания и согласование деталей требований к системе.

*Проектирование архитектурное* — определение структурных особенностей строящейся системы.

*Проектирование техническое* — отображение требований среды функционирования и разработки системы путем определения всех конструктивных элементов и их композиций.

*Проектирование детальное* — определение подробностей реализации функций для заданной среды и связей между соответствующими компонентами системы.

*Реализация программной системы* — преобразования проектных решений в работающую систему (синонимы: кодирование, конструирование).

*Родовые знания* — знания относительно всех задач семейства домена, которые представляются в виде, пригодном для обеспечения решения любой задачи, относящейся к данному семейству.

*Сертификация программного продукта* — процесс для установления соответствия программной продукции (процесса или услуг) конкретному стандарту или техническим условиям со специальным знаком или свидетельством.

*Семейство прикладных систем* — множество прикладных систем с общими функциональными свойствами и управлением.

*Связь (Relationship)* — поименованная ассоциация между двумя сущностями, имеющая значение для рассматриваемой предметной области.

*Спецификация* — описание алгоритма, правил, ограничений действий объектов с учетом стандартов, критериев качества и др.

*Спиральная модель ЖЦ* — модель процессов разработки системы, с возможностью возвращаться к любому предыдущему процессу с целью переработки элементов сделанного продукта.

*Событие* — явление, которое провоцирует смену определенного состояния и переход к другому состоянию в системе.

*Состояние (домена, системы, объекта и тому подобных)* — фиксация определенных свойств на определенный момент или интервал времени.

*Статическое тестирование* — анализ и рассмотрение спецификаций компонентов на правильность представления без их выполнения на компьютере.

*Стереотип* — указатель категории элемента моделирования UML.

*Сопровождение* — выполнение реализованных в системе задач, работы по внесению в нее изменений после того, как она передана пользователям для эксплуатации.

*Структура системы* — множество элементов и отношений между ними.

*Субъект (актор)* — кто-то или что-то вне системы, что взаимодействует с системой.

*Сущность (Entity)* — реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.

*Сценарий* — конкретная последовательность действий, которая иллюстрирует поведение и выполнение экземпляра прецедента.

*Тест* — некоторая программа, предназначенная для проверки правильности ее работы и выявления в ней ошибочных ситуаций.

*Тестовые данные* — набор данных, которые готовятся на основе документов программы или спецификаций, для проверки работы программной системы.

*Тестирование* — способ семантической отладки (проверки) программы, который состоит в выполнении последовательности различных контрольных наборов тестов и сверки полученных результатов с известными заранее.

*Требование* — соглашение или договор между заказчиком и исполнителем системы относительно свойств ее функций, условий работы в заданной среде.

*Унаследованная система* — существующая действующая система, созданная с помощью любых методов и технологий для поддержки некоторых процессов бизнеса.

*Управление качеством* — комплекс способов и системной деятельности по планированию, управлению и оценке качества программного обеспечения.

*Упрятывание информации* — принятие решения о том, что следует сообщить всем о программе, а что оставить при себе — не показывать.

*Функция* — содержание действий, выполнение которых возлагается на элемент системы при заданных требованиях, условиях и ограничениях.

*Функциональные требования* — это условия и ограничения на цели, функции системы и принципы их выполнения на компьютере.

*Функциональная полнота* — атрибут, показывающий степень достаточности основных функций для решения специальных задач соответственно назначению программного обеспечения.

*Функциональная структура* — структура, элементами которой являются функции, реализуемые подразделениями предприятия, а отношениями — связи, обеспечивающие передачу предметов труда.

*Характеристики качества* — функциональность (functionality), надежность (realibility), удобство (usability), эффективность (efficiency), сопровождаемость (maitainnability), переносимость (portability).

*Черного ящика метод* — тестирование реализованных функций путем проверки соответствия реального поведения функций с ожидаемым поведением исходя из спецификаций требований.

*Экземпляризация* — зависимость между параметризованным абстрактным классом-шаблоном (template) и реальным классом через определение параметров шаблона.

*Эксплуатация* — действия по выполнению готовой программной системы.

*UML* (Unified Modeling Language) — диаграммный способ (язык) для спецификации, визуализации, конструирования и документирования продуктов на процессах ЖЦ.

Учебное издание

**Катаев** Михаил Юрьевич

ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

Учебное пособие

Корректор Осипова Е. А.
Компьютерная верстка Риб Е. О.

Подписано в печать 11.04.13 Формат 60x84/8.
Усл. печ. л. 18,6. Тираж 300 экз. Заказ

---

Издано в ООО «Эль Контент»
634029, г. Томск, ул. Кузнецова д. 11 оф. 17
Отпечатано в Томском государственном университете
систем управления и радиоэлектроники.
634050, г. Томск, пр. Ленина, 40
Тел. (3822) 533018.