Министерство науки и высшего образования Российской Федерации
Сибирский государственный университет науки и технологий
имени академика М. Ф. Решетнева

Г. А. Доррер

МЕТОДОЛОГИЯ ПРОГРАММНОЙ ИНЖЕНЕРИИ

*Утверждено редакционно-издательским советом университета
в качестве учебного пособия для студентов магистратуры
по направлению подготовки 09.04.04 «Программная инженерия»
всех форм обучения*

Красноярск 2021

УДК 681.3.06(075.8)
ББК 32.97я73
Д69

Рецензенты:
доктор технических наук, профессор С. В. Ченцов
(Сибирский федеральный университет);
доктор технических наук, профессор А. В. Мурыгин
(Сибирский государственный университет науки и технологий
имени академика М. Ф. Решетнева)

Доррер, Г. А.

Д69 Методология программной инженерии : учеб. пособие / Г. А. Доррер ; СибГУ им. М. Ф. Решетнева. – Красноярск, 2021. – 190 с.

Описаны этапы становления системной и программной инженерии, основные черты методологии программной инженерии, международные своды знаний по системной инженерии SEBOK и по программной инженерии SWEBOK. Приводятся сведения о системах, системном анализе и методологии.

Рассмотрены практические вопросы реализации жизненного цикла программных систем, а также отечественные и международные стандарты, связанные с программной инженерией – Государственный образовательный стандарт по направлению подготовки 09.04.04 «Программная инженерия», профессиональные стандарты, а также стандарты, определяющие процессы жизненного цикла программных систем.

Предназначено для студентов магистратуры по направлению подготовки 09.04.04 «Программная инженерия» при изучении дисциплины «Методология программной инженерии». Кроме того, оно может быть полезным студентам других направлений и специальностей при ознакомлении с основами системной и программной инженерии.

УДК 681.3.06(075.8)
ББК 32.97я73

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Программная инженерия (Software Engineering) – сравнительно новое актуальное направление научной и инженерной деятельности, сформировавшееся в последние десятилетия. В центре внимания этого направления находятся вопросы научного планирования, проектирования, оценки, конструирования и эффективного использования крупных программных систем, а также проблемы успешной организации коллективных, бригадных методов работы при создании и функционировании таких систем.

Программная инженерия – весьма обширная область знаний и технологий. Учебный курс «Методология программной инженерии» предназначен для первоначального ознакомления студентов с методами и технологиями разработки и эксплуатации больших программных систем, с профессиональными требованиями к специалисту по программной инженерии и их отличиями от требований к традиционному специалисту по Computer Science.

Уделяется внимание моделям управления жизненным циклом сложных программных систем и технологиям поддержки процессов реализации всех этапов жизненного цикла.

Предполагается, что студенты-магистранты, ранее освоившие бакалаврскую программу по данному направлению, знакомы с информатикой, программированием на языках высокого уровня, базами данных, web-технологиями, объектно-ориентированным моделированием и программированием, в том числе с языком UML, технологиями, CASE и IDE – средствами разработки программного обеспечения.

Книга содержит 6 глав.

В первой главе кратко рассматриваются предпосылки возникновения и становления системной и программной инженерии как отдельных инженерных дисциплин.

Вторая глава посвящена рассмотрению системных понятий и определений, необходимых для изложения дальнейшего материала. Дается понятие системы, системного анализа, методологии, моделирования «мягких» систем.

В третьей главе рассмотрены основные черты системной и программной инженерии и методологии создания больших систем. Кратко рассмотрен также свод знаний по системной инженерии SEBOK (System Engineering Body of Knowledge).

Четвертая глава посвящена знакомству со сводом знаний по программной инженерии SWEBOK (Software Engineering Body of Knowledge).

В пятой главе описаны практические аспекты реализации всех этапов жизненного цикла программных проектов.

Шестая глава посвящена описанию стандартов, определяющих как разработку программных систем, так и подготовку специалистов по программной инженерии. Рассмотрен образовательный стандарт по направлению 09.04.04, ряд профессиональных стандартов и стандартов по управлению жизненным циклом программного обеспечения.

Изучение курса «Методология программной инженерии» предполагает практические и самостоятельные занятия, знакомство с литературой и нормативными документами по программной инженерии, в том числе, на английском языке, выполнение заданий и курсовой работы, а также прохождение практик в организациях, занятых разработкой сложных программных систем.

Пособие предназначено для изучения дисциплины «Методология программной инженерии», студентами-магистрантами, обучающимся по направлению 09.04.04 «Программная инженерия».

В пособии использован материал, содержащийся в ряде учебников и учебных пособий [10–13; 16–18; 20; 21], а также в стандартах и сводах знаний по системной и программной инженерии SEBOK [25] и SWEBOK [26].

Глава 1

О СТАНОВЛЕНИИ СИСТЕМНОЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

В середине XX века успехи науки, техники и технологий наряду с быстро возраставшими потребностями в автоматизации процессов и производств на основе стремительно совершенствовавшихся компьютеров стимулировали начало индустриального создания так называемых систем «большого масштаба».

Эти системы отличаются как количественными показателями:

– существенно возросло число составных частей и выполняемых функций;

– качественно повысилась степень автоматизации;

– заметно повысились стоимость систем и важность решаемых ими задач;

так и качественными характеристиками:

– принципиально возрос уровень организации и управления;

– усложнилось функционирование системы в целом и ее частей;

– повысилась неоднородность;

– появилась принципиальная потребность в совместной работе с другими, весьма сложными системами.

В основу работ по созданию систем «большого масштаба» легли достижения общей теории систем, системного анализа, исследования операций, теории оптимизации, вычислительной техники и программного обеспечения. Эти достижения стали целенаправленно использоваться при комплексном решении инженерных и организационно-управленческих задач, возникающих при создании подобных систем, что в итоге привело к появлению нового междисциплинарного подхода и методики, получившего название ***системная инженерия – System Engineering*.** В центре внимания системной инженерии оказались вопросы научного планирования, проектирования, оценки, конструирования и эффективного использования систем, создаваемых людьми для удовлетворения установленных потребностей, а также проблемы успешной организации коллективных, бригадных методов работы при создании таких систем.

Системная инженерия на основе объединения достижений различных дисциплин и групп специальностей предоставила методологический базис и средства для успешной реализации согласованных, командных усилий по формированию и реализации хорошо структурированной деятельности по созданию систем различных классов, от-

вечающих установленным требованиям, деятельности, которая охватывает все стадии жизненного цикла системы – от замысла до изготовления, эксплуатации, модификации и прекращения применения.

Одними из первых в середине 50-х годов комплекс проблем и подходов системной инженерии рассмотрели Г. Гуд и Р. Макол в своей книге «Системотехника. Введение в проектирование больших систем». В частности, авторы в своей книге отмечали, что создаваемые людьми большие сложные системы отличаются следующими чертами:

– целостностью, или единством системы: у всей системы имеются какие-то общие цели, общее назначение;

– большими размерами систем – как по числу частей, числу выполняемых функций, числу входов, так и по своей стоимости;

– сложностью поведения системы, например, тем, что изменение одного параметра может повлечь за собой изменение многих других параметров, характеризующих и поведение, и состояние системы;

– высокой степенью автоматизации, что позволяет решать не только технические, но и управленческие задачи;

– нерегулярностью поступления внешних возмущений – с вытекающей отсюда невозможностью точного предсказания нагрузки;

– наличием (в большинстве случаев) в составе системы состязательных конкурирующих сторон;

– усилением внимания к возможностям и функционированию человека-оператора и существенным повышением роли эффективной организации взаимодействия «человек-машина»;

– повышением требований к использованию адекватных методов, облегчающих принятие решений персоналом;

– появлением новых способов организации деятельности по созданию систем с особым акцентом на коллективные, бригадные методы работы.

Реализация таких систем потребовала использования ЭВМ и программного обеспечения.

Программное обеспечение (ПО) – это набор компьютерных программ, процедур и связанной с ними документации и данных (определение стандарта ISO/IEC 12207).

Взгляд на ПО только как на программу, загруженную в компьютер, слишком узок. Дело в том, что разрабатывается (поставляется) не только программа, но еще и документация, в которой можно прочитать как установить программу и как ей пользоваться и данные для установки программы в различных условиях (конфигурационные файлы).

Поэтому ПО иногда называют программным продуктом. Иными словами, программный продукт (программное обеспечение) – это не только программы, а также вся связанная с ними документация и конфигурационные данные, необходимые для корректной работы программы.

В зависимости от того, для кого разрабатываются программные продукты (конкретного заказчика или рынка), программные продукты бывают двух типов:

– коробочные продукты (*generic products* – общие продукты или *shrink-wrapped software* – упакованное ПО);

– заказные продукты (*bespoke* – сделанный на заказ или *customized products* – настроенный продукт). Важная разница между ними заключается в том, кто ставит задачу (определяет, или специфицирует требования) и кто несет ответственность за качество продукта.

В конце 60-х годов прошлого века в ряде стран было отмечено явление под названием «*software crisis*» (кризис ПО).

Это выражалось в том, что большие проекты стали выполняться с отставанием от графика или с превышением сметы расходов, разработанный продукт не обладал требуемыми функциональными возможностями, производительность его была низка, качество получаемого программного обеспечения не устраивало потребителей.

Это явление казалось особенно странным на фоне стремительного развития аппаратного обеспечения, когда производительность компьютеров увеличивалась вдвое каждые два года. Казалось, что и в разработке программного обеспечения должен произойти прорыв, позволяющий легко и быстро создавать сложные программные системы. Однако такого прорыва до сих пор не произошло, и по мнению специалистов, он вряд ли произойдет – не появятся изобретения, способные повлиять на продуктивность создания, надежность и простоту программного обеспечения так, как электроника, транзисторы интегральные схемы повлияли на аппаратное обеспечение компьютеров. Не следует ожидать, что когда-либо в будущем каждые два года будет происходить двукратный рост производительности труда программистов.

Объективная потребность управлять процессом разработки сложных систем ПО, прогнозировать и гарантировать стоимость разработки, сроки и качество результатов привела в конце 60-х годов прошлого века к необходимости перехода от кустарных к индустриальным способам создания ПО и появлению совокупности инженер-

ных методов и средств создания ПО, объединенных общим названием ***программная инженерия – software engineering и CASE – Computer-Aided Software Engineering*** – набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов. Понятие *CASE* используется в настоящее время в весьма широком смысле. Первоначальное значение этого понятия, ограниченное только задачами автоматизации разработки ПО, в настоящее время приобрело новый смысл, охватывающий большинство процессов жизненного цикла ПО. Эти методы и технологии особенно быстрыми темпами развивались в 60–70-х годах в интересах аэрокосмической и оборонной отраслей промышленности. Суть упомянутых методов состоит в применении систематизированного, основанного на системном анализе подхода к принятию решений, обеспечивающих эффективный переход от концепции системы к пригодным для успешной реализации проектным решениям и в конечном счете к пригодной для использования системной продукции.

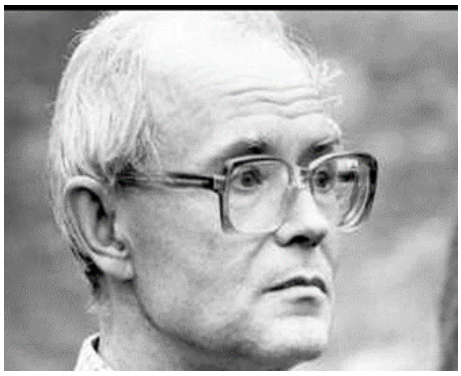
Новое направление развивалось усилиями многих специалистов – ученых и программистов. Отметим специалистов, которые внесли важный вклад в становление программной инженерии.

Липаев Владимир Васильевич

В нашей стране одним из ведущих разработчиков больших программных систем и теоретиков программной инженерии явился доктор технических наук, профессор **Владимир Васильевич Липаев** (1928–2015), руководивший в 1960-е – 1980-е годы созданием многих программных систем оборонного назначения. В частности, он был главным конструктором систем, созданных в период с 1979 по 1989 годы в рамках проекта ПРОМЕТЕЙ (ПРОектирование Математики Единая Технология) для исследования, создания и внедрения автоматизированной технологии разработки программного обеспечения комплексов, работающих в реальном времени на специализированных ЭВМ в оборонной промышленности. Ставилась задача создать технологии и среду разработки, которая обладала бы высоким качеством, была надежна и безопасна при производстве программ

реального времени для систем оборонного назначения. Такая инструментальная среда (система) должна была быть мобильной и применяться на широком классе специализированных и универсальных отечественных ЭВМ (БЭСМ, М-20 и др.). Основная цель состояла в том, чтобы на специализированных ЭВМ создать программное обеспечение, основанное на общих принципах разработки, которые используются на универсальных ЭВМ. Для этих целей в рамках проекта ПРОМЕТЕЙ-технологии были созданы отдельные системы ЯУЗА, ТЕМП, РУЗА, ДВИНА, ПРОТВА, ОХТА, ПОМПА, ПРА, НАРА.

В последующие годы В. В. Липаев возглавлял научно-технический Совет «Информатизация России», который разработал концепцию информатизации нашей страны. Им написано множество монографий и учебников по программной инженерии. Эти книги наряду с другими источниками использованы в нашем курсе.



Ершов Андрей Петрович

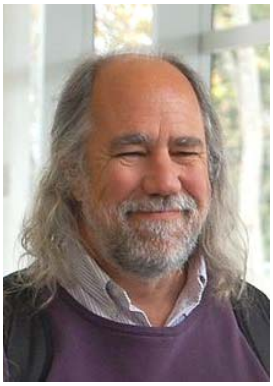
Большой вклад в развитие основ программной инженерии внес **Андрей Петрович Ершов** (1931–1988) – один из пионеров теоретического и системного программирования, создатель Сибирской школы информатики, академик АН СССР. Его работы оказали огромное влияние на формирование и развитие вычислительной техники не только в СССР, но и во всём мире. Под его руководством и при его участии были созданы такие Алголо-подобные языки программирования, как Альфа, Альфа-6 и трансляторы с них. В 1970-е годы он разработал типовую, общую для многих языков схему трансляции, пригодную для создания фрагментов оптимизированных трансляторов. С 1966 по 1972 год руководил созданием программно-аппаратной системы разделения времени АИСТ («автоматическая информационная станция») в СО АН СССР. Труды А. П. Ершова по информатике, в том числе по теоретическому и системному программированию, получили международное признание: он был членом Ассоциации вычислительной техники, почётным членом Британского общества по вычислительной технике.

А. П. Ершов многое сделал для развития школьной информатики в нашей стране и во всем мире. В 1981 году на 3-й Всемирной конференции Международной федерации по обработке информации и

ЮНЕСКО по применению ЭВМ в обучении (Лозанна, Швейцария) он сделал доклад под названием «Программирование – вторая грамотность». Название доклада быстро стало лозунгом. С тех пор началось преподавание информатики как учебного предмета во многих школах Советского Союза, а впоследствии – России.

В 80-х и 90-х годах как ответ на кризис ПО произошли большие изменения в методологии и технологии разработки программных систем, связанные с широким использованием объектно-ориентированного подхода к проектированию сложных программных систем.

Одним из главных инструментов этой технологии стал унифицированный язык моделирования UML (Unified Modeling Language). В разработке этого подхода приняли участие многие специалисты, среди которых выделяются «three amigos – трое друзей» – Греди Буч (Grady Booch), Ивар Якобсон (Ivar Jacobson), Джим Рамбо (James Rumbaugh). Этими авторами написан ряд книг по программной инженерии, ставших классическими учебниками.



Греди Буч

Ивар Якобсон

Джим Рамбо

Переход к массовому созданию сложных систем большого масштаба в последние годы был существенно ускорен благодаря быстрому развитию и усложнению информационных технологий, что нашло отражение в начале промышленного создания так называемых программно-интенсивных систем, основанных на компьютерах, которые стали основой систем обобщенного предприятия и, наконец, беспрецедентных по сложности суперсистем, получивших название ***систем систем – Systems of System.***

Беспрецедентный рост совокупных затрат на создание таких объектов, достигающий в мировых масштабах многих миллиардов долларов, необходимость постоянного продления жизненного цикла

(ЖЦ) ранее построенных систем привлекли усиленное внимание к такому разделу системной инженерии, как ***управление жизненным циклом систем.*** В рамках этого раздела системная инженерия дает методы и средства выбора, адаптации и практического использования фундаментальной совокупности процессов, охватывающих все аспекты ЖЦ сложных рукотворных систем практически любого назначения. На основе этих процессов возможны четкая организация и планирование работ на предприятии, эффективное управление проектами и ресурсами, принятие обоснованных инженерных и управленческих решений в условиях рисков, успешное управление качеством и конфигурацией, реализация всего спектра технических решений от концепции системы до вывода ее из эксплуатации. Достижения системной инженерии в области управления ЖЦ систем стали эффективным ответом на инженерные вызовы XXI века.

Особо нужно отметить сложные заказные программные продукты**,** предназначенные для систем управления и обработки информации реального времени. Этот вид программных продуктов заказывается, разрабатывается и активно применяется в крупных системах динамического управления объектами и процессами в высокоточном технологическом производстве, в авиации, в управлении космическими аппаратами, атомными электростанциями, оборонной техникой. Они являются наиболее сложными компонентами интеллектуальных систем особенно высокого качества, создаваемых человеком. При проектировании и производстве сложных заказных программных продуктов используются основные понятия, принципы и процессы современного индустриального производства сложных технических систем. Такие комплексы программы проектируются и производятся как промышленные изделия по методам, правилам и стандартам, характерным для производства сложной высококачественной технической продукции.

В результате происходящих сегодня технологических и общественных изменений сфера применения и содержание системной и программной инженерии заметно расширились. Их предметом является интегрированное, целостное рассмотрение крупномасштабных, комплексных, высокотехнологичных систем, взаимодействующих преимущественно на уровне предприятия с использованием социотехнических интерфейсов. Создание таких систем требует усиленного внимания:

– к разработке архитектуры систем, проектированию систем и их элементов,

– системному анализу и исследованию операций,
– управлению инженерной деятельностью,
– выбору технологий и методик,
– эффективному управлению ЖЦ.

При этом современная системная и программная инженерия не заменяет «традиционную», а, напротив, базируется на ее достижениях, в первую очередь в части фундаментальных процессов проектирования и разработки.

Высокая сложность современных систем существенно затрудняет использование точных, хорошо формализованных методов при их создании. Особенностью создания больших систем является необходимость учета человеческого фактора – мнения так называемых *стейкхолдеров (stakeholder)*, людей причастных к созданию или использованию систем. Работа со стейкхолдерами требует особого подхода, который называется «мягким» системным анализом. Этот подход кратко рассмотрен во второй главе пособия.

Таким образом, основные концепции, методы и технологии современной системной и программной инженерии сформировались главным образом в рамках практики успешных разработок.

Международным профессиональным сообществом созданы организации, которые проводят работу по стандартизации в области системной и программной инженерии по согласованным программам. В частности, к наиболее авторитетным организациям относятся следующие.

1. Объединенный технический комитет ISO (International Standard Organization) и IEC (International Electrotechnical Commission), где рассматриваемыми вопросами занят в первую очередь 7-й подкомитет «Программная и системная инженерия» (ISO/IEC/JTC1/SC7).

2. Институт инженеров электротехники и электроники (Institute of Electrical and Electronics Engineer – IEEE).

3. Международный совет по системной инженерии (International Council on Systems Engineering – INCOSE).

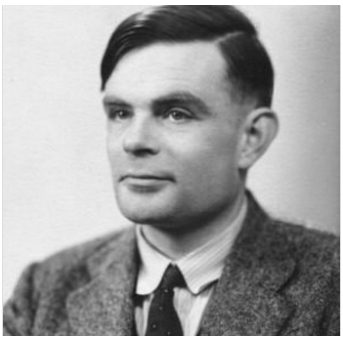
Кроме того, существенный вклад в разработку нормативно-технической базы системной инженерии внесли Альянс электронной индустрии (Electronic Industries Alliance – EIA), Институт программной инженерии университета Карнеги-Меллон (Software Engineering Institute Carnegie Mellon University – SEI CMU) и ряд других имеющих международное признание организаций.

В Российской Федерации созданием Национальных стандартов занимается Федеральное агентство по техническому регулированию и метрологии, которое сотрудничает с упомянутыми выше международными организациями.

Большую роль в становлении программной инженерии играют специализированные журналы и конференции, которые способствуют обмену результатами и образованию профессионального сообщества в данной области.

Из зарубежных журналов отметим «ACM Transactions of Software and Methodology» и «IEEE Transactions of Software Engineering», а из отечественных – «Открытые системы» и «Программная инженерия».

Из регулярно проводимых конференций можно назвать Международный симпозиум «ACM SIGSOFT on the Foundation of Software Engineering».



Алан Тьюринг

Для поощрения выдающихся работ в области информатики, в том числе и программного обеспечения, установлена Премия Тьюринга (*Turing Award*) – самая престижная премия, установленная в память о гениальном ученом Алане Тьюринге (1912–1954) и вручаемая Международной ассоциацией вычислительной техники за выдающийся научно-технический вклад в этой области.

Ключевую роль при реализации сложных программных проектов играет квалификация разработчиков, и этому вопросу уделяется особое внимание.

Профессионалы в области программной инженерии должны обладать большим объемом знаний. В помощь им разработан международно-признанный свод знаний по программной инженерии SWEBOK (Software Engineering Body of Knowledge), который будет рассмотрен в четвертой главе настоящего пособия.

Не менее важны практические аспекты работы программного инженера, которые приобретаются при разработке сложных систем. Комплекс вопросов, связанных с практикой разработки программных систем, мы рассмотрим в пятой главе пособия.

Программные инженеры, включая практиков, преподавателей, менеджеров и руководителей высшего звена должны добиваться, чтобы анализ, спецификация, проектирование, разработка, тестирование и сопровождение программного обеспечения стали полезной и ува-

жаемой профессией. Для этой цели профессиональными сообществами разработан этический кодекс «Software Engineering Code of Ethics and Professional Practices» (Кодекс этики и профессиональной практики программной инженерии).

Этот кодекс выглядит следующим образом.

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 соответствовал 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 в соответствии 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

В мировой практике промышленного производства компетентность организации определяется наличием стандартов на производство продуктов и услуг и сертификацией производителей на соответствие этим стандартам. Процесс стандартизации и сертификации давно вошел и в программную инженерию, где он составляет основу промышленного производства программных продуктов. Поэтому в пятой и шестой главах нашего курса процессам стандартизации, аккредитации и лицензирования будет уделено отдельное внимание.

Итак, мы видим, что в течение последних пятидесяти лет разработка программного обеспечения превратилась из отдельных по-

пыток достичь успеха, в инженерную профессию, которая характеризуется:

1) профессиональным сообществом;

2) стандартами, определяющими общепринятые профессиональные практики;

3) этическим кодексом;

4) профессиональными конференциями;

5) наличием учебников,

6) принципами обучения и учебными программами;

7) критериями и программами аккредитации,

8) сертификацией и лицензированием программ, и, наконец,

9) руководством к своду знаний SWEBOK.

При изучении предлагаемой дисциплины важно понимать, что программная инженерия мало касается вопросов конкретизации применения тех или иных языков программирования, архитектурных решений или рекомендаций, касающихся распространенных, или развивающихся технологий (например, web-служб, облачных технологий). Специалист по программной инженерии должен, конечно, обладать традиционным объемом знаний в области computer science (математика, информатика, программное обеспечение, информационные системы и др.) однако основная направленность его подготовки – технологии создания больших программных систем и управление их жизненным циклом.

Впрочем, не следует думать, что принципами и технологиями системной и программной инженерии в настоящее время овладели и пользуются все разработчики программных систем. Как показал анализ производственного опыта более 100 компаний, 50 % продуктов созданы без применения методов разработки требований; 75 % созданы без использования инструментов разработки требований; 25 % разработаны вообще без применения методов проектирования и т. д. Промышленность предпочитает эволюционное развитие революционным скачкам, тем не менее методология и технологии программной инженерии получает все большее распространение, и им, несомненно, принадлежит будущее.

Этому способствует решительный поворот в сторону цифровизации всех сфер жизни страны, о чем свидетельствует принятие в 2017 году Стратегия развития информационного общества в Российской Федерации на 2017–2030 годы, утвержденной указом Президента России от 9 мая 2017 года № 203.

Контрольные вопросы и задания

1. Назовите лауреатов премии Тьюринга (по годам) и сделайте реферат об их достижениях.

2. Прочитайте книгу Фредерика Брукса «Мифический человеко-месяц, или как создаются программные системы» [9]. Какие выводы Вы можете сделать для себя?

3. Назовите этапы эволюции сложных программных систем.

4. Дайте определение понятию «жизненный цикл программного обеспечения».

5. Перечислите основные этапы жизненного цикла ПО. Каково их назначение?

6. В чем разница между программной инженерией и системной инженерией?

7. В чем отличие программной инженерии от инженерии в других областях?

8. Как вы понимаете управление знаниями в процессе разработки программных систем?

9. Что такое CASE технологии разработки программных систем?

10. В чем заключаются бизнес-аспекты разработки программных систем?

11. Для чего нужен этический кодекс программной инженерии? Прокомментируйте все пункты этого кодекса.

Глава 2

ПОНЯТИЯ СИСТЕМЫ, СИСТЕМНОГО АНАЛИЗА, МЕТОДОЛОГИИ

2.1. Что такое система

В настоящей главе будут рассмотрены понятия и определения, необходимые для изложения дальнейшего материала. Поскольку программная инженерия порождена быстрым развитием системной инженерии и является ее частью, целесообразно начать рассмотрение вопросов методологии программной инженерии со знакомства с терминами и понятиями, лежащими в основе системной инженерии: система, системный подход, системное мышление, системный анализ, методология и другими.

Рассмотрим ряд определений.

Элемент – это минимальный неделимый объект. Элемент можно использовать только как целое, поэтому недопустимо говорить о половине или четверти элемента. Неделимость элемента – это, прежде всего, удобное понятие, но не физическое свойство. Оперируя понятием «элемент», исследователь оставляет за собой право перейти на другой уровень рассмотрения вопросов и говорить о том, из чего состоит элемент, а это свидетельствует о физической разложимости последнего. Таким образом, объекты называются элементами по соглашению, принимаемому с целью дать ответ на определенные вопросы, стоящие перед исследователями. Изменение вопросов может потребовать разложения элементов на составные части или объединения нескольких элементов в один.

Система согласно стандарту ISO/IEC15288 – это совокупность взаимодействующих элементов, организованная для достижения одной или нескольких установленных целей. Здесь под *целью* понимается совокупность результатов, определяемых назначением системы. Наличие цели и заставляет связывать элементы в систему.

Целостность – наиболее важное свойство системы. Элемент принадлежит системе потому, что он связан с другими ее элементами, так что множество элементов, составляющих систему, невозможно разбить на два и более несвязанных подмножества. Удаление из системы элемента или совокупности элементов непременно изменяет ее свойства в направлении, отличном от цели.

В ряде случаев система существует в некоторой внешней среде, тогда можно говорить о ***границе между системой и остальной***

средой. Установление границы и, следовательно, выделение системы – в конечном счете, выбор наблюдателя. Это подчеркивает тот факт, что любое конкретное определение системы представляет собой мысленную конструкцию, используемую для описания действительности.

Выделяют ***замкнутые и открытые системы***. В замкнутых системах все взаимодействия между элементами определены внутри их границ. В открытых системах помимо взаимодействия с элементами внутри границ возможно взаимодействие с элементами внешней среды.

В широком смысле термин «система» может означать инженерную систему, естественную систему, социальную систему или все три.

При этом в сложных инженерных системах нужно принимать во внимание как природные, так и социотехнические системы. П. Чекландом, введено понятие ***«мягких» систем (Soft Systems)*** чтобы выразить важность социальных факторов при решении инженерных проблем. Область охвата Soft Systems в значительной степени определяет, достигают ли инженерные системы своих целей, не оказывая отрицательного влияния на другие стороны жизни, когда эти системы развернуты в реальном мире.

Искусственные (инженерные) системы описывают путем определения их функций и структур.

Функция системы – это правило получения результатов, предписанных целью (назначением) системы. Определяя функцию системы, ее поведение описывают с использованием некоторой системы понятий – отношений между переменными, векторами, множествами и т. п. Функция устанавливает, что делает система для достижения поставленной цели безотносительно к физическим средствам (элементам, связям), составляющим саму систему, и не определяет, как устроена система. Системы изучают на разных уровнях абстракции, с использованием различных подходов, каждый из которых дает ответ на определенные вопросы. В связи с этим функции системы могут описываться с разной степенью детализации. Для описания функций систем используются различные теории: теория множеств, теория алгоритмов, теория случайных процессов, теория информации и др.

Функционировать – реализовать функцию, т. е. получить результаты, предписанные назначением системы.

Обратная связь – воздействие результатов функционирования системы на характер этого функционирования. Различают положительную и отрицательную обратную связь.

Структура системы – это фиксированная совокупность элементов и связей между ними. В общей теории систем под структурой принято понимать только множество связей между элементами, т. е. структура – картина, отображающая только конфигурацию системы безотносительно к составляющим ее элементам. Такое толкование этого понятия удобно при структурном подходе к изучению свойств различных систем – систем с параллельными, последовательными, иерархическими структурами, обратными связями и т. п. На практике в понятие «структура» включают не только множество связей, но и множество элементов, между которыми существуют связи. Этот смысл отражен в данном определении структуры. Наиболее часто структура системы изображается в форме графа: элементы системы представляются вершинами графа, а связи – дугами (ребрами) графа.

Граф – это математическая форма отображения структур. Инженерной формой изображения структур систем являются схемы. Схема и граф – понятия, адекватные по содержанию, но различные по форме. В *схемах* элементы и связи обозначаются любыми фигурами, удобными для инженерных (производственных) применений.

Организация – это способ реализации определенных функций в системах, состоящих из большого числа элементов. Обычно к одной и той же цели можно прийти различными способами, исходя из различных принципов организации систем. Каждый принцип организации задает определенный способ построения множества систем, аналогичных по назначению, но различных по функциям и структурам. Конкретная система представляет собой лишь пример реализации некоторого способа организации. Например, подавляющее большинство современных ЭВМ строится на основе одного принципа организации – принципа программного управления реализацией алгоритма на основе команд, имеющих операционно-адресную структуру. Таким образом, организация – понятие более высокого ранга, чем функция и структура; организация – это модель, на основе которой могут строиться многие конкретные системы.

Если речь идет о способе порождения функций, достаточных для достижения определенной цели (определенных результатов), то используется термин *функциональная организация.* Если же речь идет о наборе элементов и способе их соединения в структуру, обеспечивающую реализацию функций определенного класса, то используется термин *структурная организация*. Определяя некоторый способ функциональной организации, выявляют класс функций, присущих

системам определенного назначения (безотносительно к средствам, необходимым для реализации этих функций), а определяя способ структурной организации, выявляют правило построения структур, реализующих некоторый класс функций, т. е. отвечающих некоторому назначению.

Кроме того, под организацией понимают также ***совокупность процессов или действий, ведущих к образованию и совершенствованию взаимосвязей между частями системы.***

Целостность системы – ее относительная независимость от среды и других аналогичных систем.

Эмерджентность – свойство несводимости системы к свойствам отдельных элементов системы. Это одно из важнейших свойств, оно означает, что элементы, соединенные в систему, приобретают новые свойства. В программной инженерии это свойство проявляется на этапе жизненного цикла, связанном со сборкой системы из отдельных компонентов.

Анализ – это процесс определения свойств, присущих системе. Типичная задача анализа состоит в следующем. Известны функции и характеристики элементов, входящих в состав системы, и установлена структура системы. Необходимо определить функции или характеристики, присущие совокупности элементов в целом.

Синтез – это процесс порождения функций и структур, необходимых и достаточных для получения определенных результатов. Выявляя функции, реализуемые системой, определяют некоторую абстрактную систему, о которой известно только то, что она будет делать. В связи с этим этап синтеза функций называется *абстрактным синтезом*, а этап порождения структуры, реализующей заданные функции, – *структурным синтезом.*

Эффективность – это степень соответствия системы своему назначению. Из двух систем более эффективной считается та, которая лучше соответствует своему назначению. Оценка эффективности системы – одна из задач анализа систем. *Показатель эффективности* (качества) – это мера одного свойства (характеристики) системы. Показатель эффективности всегда имеет количественный смысл, т. е. является измерением некоторого свойства. По этой причине использование какого-либо показателя эффективности предполагает наличие способа измерения (оценки) значения этого показателя. Для оценок эффективности систем могут применяться, например, такие показатели, как производительность, стоимость, надежность, габариты и т. п.

Критерий эффективности – это мера эффективности системы. Критерий эффективности имеет количественный смысл и измеряет степень эффективности системы, обобщая все ее свойства в одной оценке – значении критерия эффективности. Эффективность систем, создаваемых для одной цели, оценивается на основе одного критерия, общего для этого класса систем. Различие в назначениях систем предполагает, что для оценки эффективности таких систем используются различные критерии. Если при увеличении эффективности значение критерия возрастает, то критерий называется *прямым,* если значение критерия уменьшается, то *инверсным.* Из двух систем более эффективной считается та, которой соответствует большее значение прямого критерия (меньшее значение инверсного критерия).

Результативность – способность системы достигать целевого результата. Этот показатель дополняет предыдущий, поскольку система может быть эффективной, то есть соответствовать своему назначению, но не достигать конкретных результатов функционирования.

Системный анализ (системная наука) – совокупность дисциплин, которые образуют новое научное направление на основе применения системного мышление и научного подхода к решению задач, возникающих в разных прикладных областях. Это научный метод познания, представляющий собой последовательность действий по установлению структурных связей между переменными или постоянными элементами исследуемой системы. Опирается на комплекс общенаучных, экспериментальных, естественнонаучных, статистических, математических методов. Системный анализ возник в эпоху разработки компьютерной техники. Успех его применения при решении сложных задач во многом определяется современными возможностями информационных технологий. Н. Н. Моисеев приводит довольно узкое, по его выражению, определение системного анализа: «Системный анализ – это совокупность методов, основанных на использовании ЭВМ и ориентированных на исследование сложных систем – технических, экономических, экологических и т. д. Результатом системных исследований является, как правило, выбор вполне определенной альтернативы: плана развития региона, параметров конструкции и т. д. Поэтому истоки системного анализа, его методические концепции лежат в тех дисциплинах, которые занимаются проблемами принятия решений: исследование операций и общая теория управления».

Традиционно к системному анализу относят следующие разделы науки:

1. Построение моделей систем.
2. Имитационное моделирование.
3. Теория подобия.
4. Теория управления.
5. Теория проведения экспериментов.
6. Обработка экспериментальных данных.
7. Математическое программирование.
8. Теория массового обслуживания.
9. Численные методы решения задач.
10. Методы принятия решений,

а также множество других разделов знаний, которые реализуют системный подход.

Системное мышление – способ понимания сложных ситуаций, рассматривая их как комбинации систем оно представляет собой процесс открытия и установления причин – проникновения в существо важнейших процессов, лежащих в основе проблем, с которыми мы сталкиваемся и возможностей, которые мы имеем. Основная идея системного мышления состоит в выявлении, наблюдении и понимании сложных эмерджентных поведений, возникающих в результате динамических взаимодействий нескольких систем в процессе работы.

Системное мышление может считаться неотъемлемой частью науки о системах. В отличие от более специализированных научных дисциплин, стремящихся выявить принципы и создать теорию, наука о системах развивалась благодаря вкладу различных дисциплин, в том числе, указанных выше.

С системным мышлением связано понятие ***парадокса***. Парадокс – это очевидное противоречие. Тем не менее, предметы не всегда таковы, какими они кажутся. Парадокс можно объяснить, но только пытаясь обрести мудрость свыше; для системного человека это означает взгляд вверх и наружу, а не только вниз и внутрь. Парадоксальное мышление является системным мышлением в его лучшем проявлении.

Системный подход – направление философии и методологии науки, специального научного познания и социальной практики, в основе которого лежит исследование объектов как систем. Системный подход ориентирует исследование на раскрытие целостности объекта и обеспечивающих ее механизмов, на выявление многообразных типов связей сложного объекта и сведение их в единую теоретическую картину. С практической точки зрения это способ решения реальных проблем, в котором используются инструменты системной науки, позволяющие разрабатывать и использовать системы.

2.2. Что такое методология

Методология – это учение об организации деятельности. Организовать деятельность означает упорядочить ее в целостную систему с четко определенными характеристиками, логической структурой и процессом ее осуществления – временной структурой. Не всякая деятельность нуждается в организации, в применении методологии. Как известно, человеческая деятельность может разделяться на деятельность *репродуктивную* и *продуктивную*.

Репродуктивная деятельность является слепком, копией с деятельности другого человека, либо копией своей собственной деятельности, освоенной в предшествующем опыте. Такая деятельность, как, например, однообразная деятельность рабочего на конвейере в принципе уже организована (само-организована) и, очевидно, в применении методологии не нуждается.

Другое дело – продуктивная деятельность, направленная на получение объективно или субъективно нового результата. Любая научно-исследовательская деятельность всегда направлена на объективно новый результат. Инновационная деятельность специалиста-практика может быть направлена как на объективно новый, так и на субъективно новый (для данного специалиста или для данного предприятия, учреждения) результат. Учебная деятельность всегда направлена на субъективно новый (для каждого конкретного обучающегося) результат. В случае продуктивной деятельности безусловно возникает необходимость ее организации, то есть возникает необходимость применения методологии.

Можно предложить следующую «схему методологии»:

1. Характеристики деятельности: особенности, принципы, условия, нормы деятельности.

2. Логическая структура деятельности: субъект, объект, предмет, формы, средства, методы, результат деятельности.

3. Временная структура деятельности: фазы, стадии, этапы деятельности.

Исторически известны разные типы культуры организации деятельности. В настоящее время преобладающим является ***проектно-технологический тип***, который состоит в том, что продуктивная деятельность человека (или организации) разбивается на отдельные завершенные циклы, которые называются *проектами*. ***Проект*** (от лат. projectus – брошенный вперед, выступающий, выдающийся вперёд) – это уникальная деятельность, имеющая начало и конец

во времени, направленная на достижение заранее определённого результата/ цели, создание определённого, уникального продукта или услуги, при заданных ограничениях по ресурсам и срокам, а также требованиям к качеству и допустимому уровню риска. Процесс осуществления деятельности, рассматривается в рамках проекта, реализуемого в определенной временной последовательности по фазам, стадиям и этапам, причем последовательность эта является общей для всех видов деятельности:

– фаза проектирования, результатом которой является построенная модель создаваемой системы и план ее реализации;

– технологическая фаза, результатом которой является реализация системы;

– рефлексивная фаза, результатом которой является оценка реализованной системы и определение необходимости либо ее дальнейшей коррекции, либо «запуска» нового проекта.

2.3. Когнитивные модели сложных систем

Как мы увидим в следующих главах, системная и программная инженерия тесно связана с моделированием, которое осуществляется на всех этапах жизненного цикла системы. Как правило, для этой цели используются различные программные средства, позволяющие формализовать и облегчить процесс моделирования. При этом особое место занимает процесс выработки требований к системе. От полного и качественного выполнения данного этапа во многом зависит успех всего проекта, а неполный или ошибочный список требований может привести к необходимости переделок или даже к провалу проекта.

При этом нужно учитывать, что современные сложные системы, как правило, являются социотехническими, т. е. в них важную роль играют взаимоотношения между людьми, а также между человеком и системой. Часто возникают ситуации, когда поведение лиц, вырабатывающих и принимающих решения не регламентировано жестко, в частности, при обсуждении требований к создаваемому проекту. В этом случае мы имеем дело с системами, которые принято называть когнитивными или «мягкими». Под «мягкой» системой обычно понимают сложную систему – либо вовсе не структурированную, либо слабоструктурированную систему социального типа, в которой главную роль играют взаимоотношения между людьми или организациями. Методология их описания и принятия решений существенно отличается от рассмотренного выше традиционного системного мыш-

ления, которое невольно навязывает рассмотрение объекта в четких системных терминах (структура – функции, цели – средства – оптимизация).

Поэтому мы в настоящем параграфе уделим внимание методам описания и анализа когнитивных, или «мягких» систем, основанных на неформализованных знаниях.

Такие методы были разработаны Черчменом, Акоффом, Чекландом, В. В. Подиновским и другими учеными.

Рассмотрим кратко предпосылки создания методологии «мягких» систем и соответствующих моделей.

Популярные в 60–70-х годах XX века подходы к изучению сложных, в том числе социальных систем, основанные на принципах классического системного анализа, нередко заканчивались неудачей. Традиционный «жесткий» системный подход оказался неадекватным при создании сложных систем, включающих в себя персонал, потому что такие системы в качестве активных элементов включают в себя индивидов и группы, которые отличаются сложным поведением, имеют собственные цели, взгляды, установки, определяющие выбор решений и действий.

Развитию методологии «мягких» систем (ММС) способствовало помимо упомянутого выше разочарования результатами «жесткого» моделирования социальных систем также изменение характера бизнес-процессов. Происходит переход от стандартизованного производства к выпуску продукции по индивидуальным заказам. Резко возрастают объемы информации, с переработкой которой не справляются даже компьютеры. Персонал организации становится все более образованным и склонным к самостоятельному принятию решений, норовит не всегда сообщать наверх полную и достоверную информацию, более того, иногда игнорирует указания начальства. Цели подсистем все чаще не совпадают с целями системы в целом.

В этих условиях модель управления организацией должна принципиально измениться, она должна стать «социо-системной». В первую очередь организация должна стать демократической – это означает, что в принятии решений, должны иметь возможность участвовать все заинтересованные лица (стейкхолдеры). При этом сам процесс планирования нередко более важен, чем его результат. Если кого-то не удается включить в число участников планирования, то его следует привлечь в качестве консультанта, но при этом важно соблюдать принцип добровольного участия.

Идеи ММС широко используются при создании сложных человеко-машинных систем, особенно на этапе выработки требований к создаваемой системе.

Красной нитью через все методологии «мягких» систем проходят требования учета мнений всех заинтересованных сторон. Действительно, взгляды, точки зрения, картины мира могут различаться. При этом взаимопонимание – понимание различий картин мира можно существенно облегчить и углубить, если удастся их визуализировать – представить в простой и наглядной форме.

Черчмен формулирует следующие базовые тезисы нового системного подхода к принятию решений, которые можно назвать «мягкими».

Системный подход начинается, когда вы первый раз смотрите на мир глазами другого человека.

Системный подход показывает, что картина мира каждого индивида весьма ограничена.

В системном подходе нет более квалифицированных и менее квалифицированных участников. Имеется в виду, что у включенных в данную проблемную ситуацию людей могут быть разные взгляды. Могут, например, затрагиваться вопросы морали, в которых трудно быть экспертом.

Принятие решения требует гарантированного участия представителей всех заинтересованных сторон. Согласование их интересов – сложный процесс, который никогда не заканчивается, но усилия разработчиков не пропадут, так как системный подход позволит им прийти к верному решению.

Этот же автор утверждает, что к успеху проекта ведет тщательное выполнение следующих основных принципов.

Оппонирование – в слабоструктурированных проблемах можно разобраться, если рассматривать их с различных точек зрения.

Участие – в процессе принятия решений должны участвовать представители всех заинтересованных сторон.

Интегративность – в процессе обсуждения различные точки зрения должны синтезироваться на более высоком уровне, что приводит к выработке общего согласованного решения.

Обучение – в результате участники процесса системного анализа начинают лучше понимать свою проблемную ситуацию.

Методология «мягких» систем предназначена для выявления различных точек зрения и постепенного достижения взаимопонимания. Именно в этом состоит ее принципиальное отличие от традици-

онного жесткого подхода, позволяющее говорить о становлении новой системной парадигмы. Поэтому ММС можно трактовать как процесс обучения коллективному принятию решений.

Развитие методологии «мягких» систем потребовало создания адекватных методов и средств поддержки моделирования таких систем.

Ниже рассмотрены два таких инструмента – когнитивные карты и онтологии. Одним из важных инструментов графического моделирования «мягких» систем служат так называемые когнитивные карты.

Когнитивная карта (Cognitive Map от *cognitio* – знание, познание) – инструмент методологии когнитивного моделирования, предназначенного для анализа и принятия решений в плохо определенных ситуациях – была предложена Аксельродом (R. Axelrod). Она основана на моделировании субъективных представлений экспертов о ситуации и включает в себя:

1) методологию структуризации ситуации;

2) модель представления знаний эксперта в виде знакового ориентированного графа (когнитивной карты) $CM = \{F, W\}$, где F – множество факторов ситуации; W – множество дуг, означающих причинно-следственные отношения между факторами ситуации, методы анализа ситуации.

В настоящее время методология когнитивного моделирования развивается в направлении совершенствования аппарата анализа и моделирования ситуации. Когнитивные карты создаются и видоизменяются в результате активного взаимодействия субъекта с другими субъектами и окружающим миром. При этом могут формироваться когнитивные карты различной степени общности, «масштаба» и организации. Это – субъективная картина, построенная в системе выбранных показателей (координат), в которых локализованы отдельные воспринимаемые предметы.

Выделяют карту-путь («дорожную карту») как последовательное представление связей между объектами по определенному маршруту, и карту-обозрение как одновременное представление взаимного расположения объектов.

Когнитивные карты относятся к тому же классу систем представления знаний, что и фреймы. Таким образом, когнитивную карту можно понимать как схематичное, упрощенное описание картины мира индивида, точнее, ее фрагмента, относящегося к данной проблемной ситуации. Поэтому с построения когнитивных карт можно начинать исследование и моделирование сложных слабоструктурированных систем для нахождения альтернатив решения и его принятия.

Когнитивная карта может быть визуализирована в виде графа, содержащего множество вершин, каждая из которых соответствует одному фактору или элементу картины мира индивида. Помеченная дуга, связывающая вершины A и B, соответствует причинно-следственной связи $A \Rightarrow B$, где A – причина, B – следствие.

Связь $A \Rightarrow B$, называется положительной (помечается знаком «+»), если увеличение A ведет к увеличению (усилению) B, а уменьшение A ведет к уменьшению B при прочих равных условиях. Знак «–» над дугой $A \Rightarrow B$, означает, что связь отрицательная, т. е. при прочих равных условиях увеличение A приводит к уменьшению (торможению) B и уменьшение A ведет к увеличению B. В ряде случаев на когнитивной карте можно, помимо знака, указывать относительную степень влияния одного фактора на другой в виде числовой оценки (например, в диапазоне от +1 до – 1).

Причинно-следственные связи можно также отображать в виде матрицы весов W, отображающей влияние каждого фактора на все остальные.

Рассмотрим простой пример когнитивной карты. Предположим, что при разработке проекта некоторой системы учитываются следующие факторы:

– $F1$ – бюджет проекта,

– $F2$ – сроки выполнения проекта,

– $F3$ – требования к качеству проекта,

– $F4$ – уровень квалификации персонала заказчика проекта,

– $F5$ – уровень квалификации персонала разработчика проекта.

Эксперт представил взаимное влияния факторов в следующем виде:

$F1$ влияет отрицательно на $F2$ и положительно на $F3$, $F4$;

$F2$ влияет положительно на $F1$, $F4$, $F5$;

$F3$ влияет положительно на $F1$, $F2$, $F5$;

$F4$ влияет отрицательно на $F1$, $F2$, $F5$;

$F5$ влияет отрицательно на $F1$, $F2$, $F4$ и положительно на $F3$.

Заменим оценки связей + и – соответственно на 1 и –1. Тогда данному графу будет соответствовать матрица:

$$W := \begin{pmatrix} 0 & -1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 & -1 \\ -1 & -1 & 1 & -1 & 0 \end{pmatrix}.$$

Следует отметить, что когнитивная карта отображает лишь факт наличия влияний факторов друг на друга. В ней не отражается ни детальный характер этих влияний, ни динамика изменения влияний в зависимости от изменения ситуации, ни временные изменения самих факторов. Учет всех этих обстоятельств требует перехода на следующий уровень структуризации информации, отображенной в когнитивной карте, т. е. к *когнитивной модели*. Когнитивное моделирование – это средство выявления закономерностей для получения теоретических и практических знаний о проблеме и формулирования на этой основе практических выводов.

Для получения когнитивной модели на основе когнитивной карты W обозначим $P(t)$ n-вектор значений факторов, определяющий текущее состояние системы, где t – дискретное время. Тогда в простейшем случае когнитивная модель объекта будет представлена в виде линейного разностного уравнения

$$P(t+1) = P(t) + W \cdot P(t) + \xi(t), \qquad (2.1)$$

где $P(0)$ – начальное условие и $\xi(t)$ – n-вектор случайных помех.

Анализ матрицы весов W позволяет судить об устойчивости процесса, описываемого данной когнитивной картой. Для этого необходимо проанализировать собственные числа матрицы W, и если все они лежат внутри единичного круга на комплексной плоскости, то система устойчива, если же хотя бы одно собственное число оказывается по модулю больше единицы, то система неустойчива, что может в ряде случаев трактоваться как ее способность к саморазвитию. В частности, для рассмотренной матрицы W максимальное собственное число равно 1,422, и это говорит об активности системы.

Когнитивное моделирование основано на сценарном подходе. Сценарий – совокупность тенденций, характеризующих ситуацию в настоящий момент, желаемых целей развития, комплекса мероприятий, воздействующих на развитие ситуации, и системы наблюдаемых параметров (факторов), иллюстрирующих поведение процессов.

Сценарий может моделироваться по трем основным направлениям.

1. Прогноз развития ситуации без всякого воздействия на процессы в ситуации – ситуация развивается сама по себе.

2. Прогноз развития ситуации при реализации выбранного комплекса решений (управляющих воздействий) – прямая задача.

3. Синтез комплекса решений для достижения необходимого изменения состояния ситуации – обратная задача.

В заключение этого раздела заметим, что создание и анализ когнитивных карт может рассматриваться, в частности, как первый этап жизненного цикла разработки системы – разработки требований к системе.

Контрольные вопросы и задания

1. Что такое система? Дайте определение. Чем система отличается от совокупности различных элементов?

2. Как идентифицировать границу между системой и внешней средой? Является ли граница частью системы?

3. Приведите примеры проявления эмерджентности в системах. Что означает эмерджентное поведение системы?

4. Может ли социальная система состоять из одного человека или требуются как минимум двое?

5. Обязательно ли система должна иметь цель?

6. По мнению некоторых ученых, социальной системой является любая группа, не обязательно состоящая из людей, например, рой пчел, стая птиц. Является ли в таком случае социальной системой сеть компьютеров?

7. Дайте определение понятию методологии. В чем различие между репродуктивной и продуктивной деятельностью?

8. Что означает термин «система систем»?

9. В науке управления существует строгое различие между эффективностью и результативностью систем и людей. Является ли программная инженерия эффективной или результативной, или и тем, и другим? Объясните. Приведите примеры.

10. Составьте когнитивную карту какой-либо известной вам системы. Оцените собственные числа матрицы W и устойчивость системы.

Глава 3

ОСНОВНЫЕ ЧЕРТЫ СИСТЕМНОЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

В настоящее время при создании сложных систем целенаправленно используются подходы, направленные на комплексное решение инженерных и организационно-управленческих задач, что в итоге привело к появлению нового междисциплинарного подхода и методики, получившего название ***системная инженерия – System Engineering,*** в рамках которой возникло направление ***программная инженерия – Software Engineering.***

В данной главе мы рассмотрим основные черты этих подходов.

3.1. Что такое системная инженерия

Краткое определение. Системная инженерия (System Engineering – SE) – это междисциплинарный подход и способы обеспечения воплощения успешных систем.

Более полное определение. Системная инженерия – это междисциплинарный подход, определяющий полный набор технических и управленческих усилий, которые требуются для того, чтобы преобразовать совокупность потребностей и ожиданий заказчика и имеющихся ограничений в эффективное решение и поддержать это решение в течение их жизненного цикла (стандарт ISO/IEC 24765).

В этих определениях следует подчеркнуть:

Успешные системы – это то, чем занимается системная инженерия. Слово «успешные» тут крайне важно и означает, что система должна удовлетворить нужды заказчиков, пользователей и других стейкхолдеров (stakeholder – это тот или те, кто затрагивается системой, или кто затрагивает систему).

Слово «системы» используется в специальном значении: это «системы» из системного подхода.

Междисциплинарный подход – системная инженерия претендует на то, что она работает со всеми остальными инженерными специальностями (впрочем, не только инженерными). «Подход» обычно означает какие-то наработки в одной предметной области, которые можно перенести на другие предметные области.

Слова «воплощение», «реализация» (realization, «перевод в реальность») означают буквально: создание материальной (из вещества и программ) успешной системы.

Более полное определение системной инженерии фокусируется на целостном и параллельном понимании жизненного цикла системы:

– нужд стейкхолдеров;

– исследования возможностей;

– документирования требований;

– синтезирования, проверки, приёмки и постепенного появления инженерных решений, в то время как в расчёт принимается полная проблема, от исследования концепции системы до вывода системы из эксплуатации;

– целостность (полнота охвата), междисциплинарность, охват всего жизненного цикла системы «от рождения до смерти» – это ключевое свойство, которое отличает системную инженерию от всех остальных инженерных дисциплин;

– параллельность выполнения самых разных практик (а не последовательность, как можно было бы подумать, прочитав их перечисление).

Как уже сказано, SE является междисциплинарным подходом и средством, позволяющим реализовать успешные системы. Успешные системы должны удовлетворять потребности своих клиентов, пользователей и других заинтересованных сторон. Некоторые ключевые элементы системной инженерии включают в себя:

Принципы и концепции, которые характеризуют систему, где система представляет собой взаимодействующую комбинацию элементов системы для достижения определенной цели (целей). Система взаимодействует со своей средой, которая может включать в себя другие системы, пользователей и окружающую среду. Элементы системы, которые составляют систему, могут включать аппаратное обеспечение, программное обеспечение, людей, информацию, методы, средства, службы и другие элементы поддержки.

Системный инженер (system engineer) – это роль, человек или группа людей, которые поддерживают этот междисциплинарный подход. В частности, системный инженер часто помогает выявлять и переводить потребности клиентов в спецификации, которые могут быть реализованы командой разработчиков системы.

Чтобы помочь реализовать успешные системы, системный инженер поддерживает набор процессов жизненного цикла, начинающихся на ранней стадии определения требований и концептуального проектирования и продолжающихся на протяжении всего жизненного цикла системы за счет ее производства, развертывания, использова-

ния и утилизации. Системный инженер должен анализировать, указывать, проектировать и проверять систему, чтобы гарантировать, что ее функциональные, интерфейсные, рабочие, физические и другие качественные характеристики, и стоимость сбалансированы для удовлетворения потребностей системы.

Системный инженер помогает обеспечить соответствие элементов системы целям целого и в конечном итоге удовлетворить потребности клиентов и других заинтересованных сторон, которые приобретут и используют систему.

Главный признак, отличающий системных инженеров от всех других инженеров: они отвечают за проект в целом, сразу во всех его деталях. Они ответственны за то, чтобы не было пропущено какой-нибудь мелочи, ведущей к провалу.

Возьмём самолёт или корабль – это огромное количество деталей из металла, пластика и других материалов, синхронно летящих высоко в небе или плывущих далеко в море. Системный инженер – это тот, кто придумал, как обеспечить их надёжное и экономичное соединение, увязав самые разные требования (грузоподъемность, расход топлива, дальность полёта или плавания, требования к аэродрому или порту, необходимость лёгкого обслуживания на берегу и в море, безопасность людей на борту, и т. д. и т. п.), при этом требования выдвигаются самыми разными людьми, представляющими самые разные профессиональные и общественные группы. Миллион деталей изготавливается и собирается в одно изделие – и самолёт летит, корабль плывет, обеспечивая комфорт пассажирам и прибыль владельцам!

Системная инженерия решает задачу преодоления инженерной сложности (которая определяется главным образом числом различных элементов, которые должны взаимодействовать друг с другом, чтобы получить целевую систему: сложные системы не помещаются ни в какую самую умную голову, требуется специальная дисциплина, чтобы собирать результаты деятельности самых разных инженеров в одно работоспособное целое).

Системный инженер – это тот, кто отвечает за успешность системы в целом, кто занимаются системной инженерией, а сама системная инженерия тем самым является профессией в классическом смысле этого слова: есть профессиональные ассоциации системных инженеров, проводятся профессиональные конференции, есть учебные курсы в системе высшего образования.

Системные инженеры должны быть техническими лидерами в инженерных коллективах. Понятие технического лидерства (technical leadership) означает помощь в организации коллективной мыслительной работы по отношению к той или иной технической идее: все участники проекта должны делать одну и ту же систему, а не разные. Системные инженеры не просто берут идеи от одних инженеров, а потом убеждают других инженеров их принять. Системные инженеры генерируют технические идеи самостоятельно. Это и правда и неправда одновременно. Правда в том, что системные инженеры – это технические лидеры. Неправда в том, что системный инженер – это один человек, «рулящий» огромными коллективами других инженеров. Как в традиционной инженерии произошло разделение на инженеров по специальности (механиков, электриков, программистов, теплотехников и т. д.), как в медицине произошло разделение врачей по разным врачебным специальностям, и врачи редко работают поодиночке, так подобное разделение уже произошло и в самой системной инженерии, так и в системной инженерии произошла специализация.

К профессии «системный инженер» нужно относиться примерно так же, как к профессии врач: с одной стороны, вы никогда не спутаете стоматолога и гинеколога, но с другой стороны, эти обе профессии врачебные. Системные инженеры так же разнообразны, как и врачи: системный инженер, занимающийся ракетами и системный инженер, занимающийся небоскрёбами, будут иметь как много различий, так и много общего.

Системного инженера иногда сравнивают с дирижером симфонического оркестра. Эта метафора соответствует административной модели управления, когда системный инженер непосредственно командует каждым инженером по специальности, указывает, когда и какую ему ноту играть, как это происходит в симфонических оркестрах. Но ведь ещё в прошлом, XX веке музыка пошла другими путями: в музыкальной мейнстримной культуре преобладает джаз, подразумевающий совсем другие принципы коллективного творчества: импровизация плюс взаимное подстраивание. Если посмотреть на то, что происходит в менеджменте и инженерии, то тренд к «джазовой» организации деятельности несомненен. Метафора, при которой все главные решения принимаются одним лицом (дирижером), опасна, она потенциально создаёт угрозу появления в проекте узкого места, «бутылочного горлышка», существенно замедляющего принятие инженерных решений. Эта метафора не соответствует духу времени.

3.2. Свод знаний по системной инженерии SEBOK

Из предыдущего материала следует, что системный инженер должен обладать большим объемом специфических знаний. Международная группа специалистов разработала документ, содержащий ***свод знаний по системной инженерии – SEBOK (System Engineering Body Of Knowledge***).

Со сводом знаний SEBOK связан еще один продукт, который использует содержание SEBOK для определения основного объема знаний (CorBOK), который должен быть включен в учебные планы SE, называемые «Учебная программа выпускников по системной инженерии» (GRCSE). GRCSE не является стандартом, а базовым учебным планом, который должен быть адаптирован и расширен, чтобы соответствовать задачам программы выпускников каждого университета. Эти продукты разрабатываются в рамках проекта «Комплекс знаний и учебная программа для развития системной инженерии» (BKCASE).

BKCASE контролируется Советом управляющих, в состав которого входят Международный совет по системному проектированию (INCOSE), Исследовательский центр системных исследований (SERC) и IEEE Computer Society. SEBOK представляет собой сборник ключевых источников знаний и ссылок System Engineering, организованных и разъясненных для оказания помощи широкому кругу пользователей. Это живой, постоянно обновляемый продукт.

Системное проектирование является междисциплинарным подходом и средством обеспечения полного жизненного цикла успешных продуктов, услуг и корпоративных систем. Оно включает обнаружение и формулировку проблемы, определение и реализацию решения, а также оперативное использование, поддержку и завершение. Методика может быть применена к одиночным проблемным ситуациям, а также к управлению несколькими процессами в коммерческих или государственных предприятиях.

Руководство SEBOK – весьма обширный документ (815 страниц англоязычного текста, набранного 10-м шрифтом) и содержит 7 основных разделов (доменов). Перечисленные ниже разделы 1–6 фокусируется на независимой от домена информации – универсальной для системной инженерии, независимо от того, в какой области она применяется.

1. Введение. Обзор роли и содержания SE.
2. Основы SE. Знания о системах и их использование в SE.

3. SE и менеджмент. «Стандартный ЖЦ», процессы и практика.

4. Применение SE. Различные контексты, в которых «Стандартный ЖЦ» используется.

5. Возможности SE. Подбор людей, команд и предприятий для обеспечения успешной SE.

6. Связанные дисциплины. Другие дисциплины, включенные в ЖЦ и как их использовать.

В седьмом разделе SEBOK приведены примеры успешно выполненных проектов:

1. Космический телескоп Хаббл.
2. Система глобального позиционирования GPS.
3. Линейный ускоритель для онкологических исследований.
4. Фактографическая файловая система ФБР.
5. Быстрая разработка миниатюрного космического аппарата.
6. Медицинский инфузионный насос нового поколения.

Также кратко рассмотрен ряд других успешных проектов.

В данном пособии нет возможности подробно рассмотреть SEBOK, однако, на практических занятиях студенты смогут ознакомиться с этими выдающимися проектами.

3.3. Основные черты методологии программной инженерии

Основные концепции, методы и технологии современной программной инженерии формировались главным образом в рамках практики успешных разработок, поскольку большая сложность современных программных систем и их социотехнический характер существенно затрудняют использование точных, хорошо формализованных методов системного анализа при их создании. Как следствие, важнейшие аспекты, связанные как собственно с современными процессами разработки систем, так и с управлением их жизненным циклом, нашли наиболее полное и формализованное отражение в комплексе официальных и фактических международных стандартов, ставших ключевым компонентом методологического базиса современной системной и программной инженерии.

Еще в 1975 г. Фредерик Брукс, проанализировав свой уникальный по тем временам опыт руководства крупнейшим проектом разработки операционной системы OS/360 для ЭВМ IBM/360, определил перечень неотъемлемых свойств ПО: сложность, согласованность, изменяемость и незримость.

Что же касается современных крупномасштабных проектов ПО, то они характеризуются, как правило, следующими особенностями.

Характеристики объекта внедрения:

– структурная сложность и территориальная распределённость;

– функциональная сложность: многоуровневая иерархия и большое количество функций, выполняемых организацией; сложные взаимосвязи между ними;

– информационная сложность: большое количество источников и потребителей информации, разнообразные формы и форматы представления информации, сложная информационная модель объекта;

– сложная динамика поведения, обусловленная высокой изменчивостью внешней и внутренней среды.

Технические характеристики проектов создания ПО:

– различная степень унификации проектных решений в рамках одного проекта;

– высокая техническая сложность, определяемая наличием совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования;

– отсутствие полных аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем, высокая доля вновь разрабатываемого ПО;

– большое количество и высокая стоимость унаследованных приложений (существующего прикладного ПО), функционирующих в различных средах, необходимость интеграции унаследованных и вновь разрабатываемых приложений.

Опыт разработки подобных проектов привел к выделению программной инженерии как самостоятельного научного и технологического направления.

Программная инженерия (Software Engineering) – это раздел системной инженерии, наука и технологии, одной из главных задач которых являются исследование и разработка методов и средств построения систем, настолько больших и сложных, что для этого требуется участие слаженных команд разработчиков различных специальностей и квалификаций. Обычно такие системы существуют и применяются долгие годы, развиваясь от версии к версии, претерпевая в своем жизненном цикле большое число изменений. Улучшаются существующие функции, добавляются новые или удаляются устаревшие возможности. Программные системы адаптируются для работы в новой среде, устраняются дефекты и ошибки.

Важнейшим требованием в системной и программной инженерии является следование определенной методологии.

Суть методологии программной инженерии состоит в применении систематизированного, научного и предсказуемого процесса проектирования, производства и сопровождения программных комплексов, в том числе, и реального времени.

Управление промышленным проектированием и производством сложных программных продуктов – это особый вид рациональной, коллективной инженерной деятельности, включающий все этапы жизненного цикла ПО:

1) определение требований;

2) постановку задач;

3) исследования и подготовку проектных решений;

4) планирование процесса производства;

5) организацию работ;

6) контроль за ходом выполнения работ и использованием ресурсов;

7) управление конфигурацией продукта.

Задачи целевого управления такими работами – сводить воедино усилия руководителей, исполнителей – специалистов разной квалификации, подрядчиков и субподрядчиков, добиваясь, чтобы они выступали как целевая «команда», а не как разрозненные группы, реализующие свои цели. В результате должна обеспечиваться концептуальная целостность создания технических систем и высокое качество решения главных задач заказчиков при сбалансированном использовании ресурсов.

Для решения задач программной инженерии формируется новое направление исследований − экономика жизненного цикла программных комплексов. Это направление объективно включается в исследования экономики промышленного производства средств и систем вычислительной техники. Объективно исследования на этом направлении осложняются трудностью измерения экономических характеристик производства программного продукта. Широкий спектр количественных и качественных показателей, которые с различных сторон характеризуют содержание подобных продуктов, и невысокая достоверность оценки их значений, определяют значительную неопределенность при попытках описать и измерить экономические свойства создаваемых или используемых крупных комплексов программ. Тем не менее, указанное направление интенсивно развивается о нем пойдет речь в четвертой и пятой главах этой книги.

Вследствие роста сфер применения и повышения ответственности резко возросла необходимость гарантировать высокое качество

программных продуктов, регламентирования и корректного формирования требований к характеристикам реальных комплексов программ и их достоверного определения. Непременным требованием к разработкам является получение работоспособной системы «с первого раза», т. е. без переделок и доделок. В результате специалисты в области теории и методов, определяющих качество продукции, вынуждены осваивать сферу развития и применения нового, несвойственного для их основной деятельности продукта – отдельных программных продуктов и систем в целом, а также подходов к оценке их качества при использовании. Сложность анализируемых объектов и уверенность отдельных программистов в собственной «непогрешимости», зачастую приводят к тому, что реальные характеристики качества функционирования программных продуктов остаются неизвестными не только для заказчиков и пользователей, но также для самих разработчиков. Отсутствие четкого декларирования в простых документах атрибутов и значений, характеризующих характеристики качества программных комплексов, вызывает конфликты между заказчиками-пользователями и разработчиками-поставщиками из-за разной трактовки одних и тех же характеристик.

Руководство организации-заказчика должно быть уверено, что ее требования полностью понятны исполнителю – поставщику и могут быть выполнены. Для этого заказчик во взаимодействии с исполнителем должен установить общесистемные процедуры для управления требованиями и документами, которые необходимы для эффективного функционирования системы проектирования и производства. Такая система должна обеспечивать уверенность заказчика в том, что документы анализируются, при необходимости, уточняются и снова утверждаются.

Управление специалистами необходимо для того, чтобы подтверждать заказчику должный уровень их компетенции для осуществления своей деятельности, который гарантирует соответствующее образование, знания и практические навыки проектирования и производства продукции.

Методы программной инженерии поддерживают и конкретизируют технологический процесс, а также обеспечивают постоянный контроль за значениями качества программного продукта на всех стадиях его производства. Для каждого проекта создания программного продукта, выполняющего ответственные функции, должны разрабатываться и применяться система качества, планы и программа испытаний, в рамках которых реализуется специальная методология

и используются определенные инструментальные средства разработки и испытаний. Такая методология и инструментарий должны обеспечивать заданные на стадии проектирования качество, надежность и безопасность функционирования программных продуктов. Эти методы и процессы позволяют разработчикам и заказчикам программных продуктов более корректно взаимодействовать *при определении и реализации требований контрактов и технических* заданий.

Для эффективного процесса изготовления определенного программного продукта и его использования в конкретных условиях эксплуатирующего предприятия, необходимо создание соответствующей технологии его проектирования и производства. Технология производства включает методы, используемые для этого оборудования, инструментальные средства и систему взаимосвязанных способов изготовления продукции и выполнения установленного вида работ. Технология должна включать весь перечень последовательных операций по превращению исходного замысла проекта, требований заказчика и потребителей, в готовый продукт, с указанием методов, типа и характеристик инструментальных средств, которыми специалисты пользуются на каждом этапе производства.

Опыт создания сложных программных систем потребовал создания новых методов проектирования и разработки программного обеспечения и большого количества инструментальных программных средств, облегчающих их разработку. Такие средства получили название CASE-систем (Computer Aided Software Engineering systems). Одним из популярных в конце 1970-х средств, получивших признание и широкое распространения, были средства структурного моделирования, появившиеся вследствие распространения в среде разработчиков структурного программирования. Однако в дальнейшем были разработаны другие методы моделирования и проектирования программных систем, основанные на объектно-ориентированном подходе.

Организация технологии проектирования и производства – система мер, направленных на рационализацию взаимодействия в пространстве и времени материальных компонентов и специалистов, занятых в процессе производства. Дифференциация коллективного труда предполагает разделение производственного процесса на отдельные части (процессы, операции) и их закрепление за соответствующими подразделениями и специалистами предприятия. В практической деятельности по организации производства, приоритет должен отдаваться тому методу, который обеспечит наилучшие экономические и социальные характеристики производственного процесса.

Основные концепции программной инженерии сконцентрированы в целостном комплексе систематизированных международных стандартов, охватывающих и регламентирующих практически все процессы жизненного цикла сложных программных средств. Несколько десятков стандартов этого комплекса допускают целенаправленный отбор процессов. С использованием положений этих стандартов формируют проблемно-ориентированные профили стандартов, предназначенные для определенных типов проектов и/ или предприятий – заказчиков. Практическое применение профилей стандартов, сосредоточивших мировой опыт создания различных типов крупных комплексов программ, способствует значительному повышению производительности труда специалистов и качества создаваемых программных продуктов. Эти стандарты определяют эффективную модифицируемость, мобильность и возможность повторного применения программных компонентов и комплексов, их расширяемость и переносимость на различные аппаратные и операционные платформы. Указанные факторы отражаются на росте экономической эффективности технологий и процессов создания различных программных средств и систем. Для регламентирования процессов жизненного цикла такие профили стандартов должны адаптироваться и конкретизироваться применительно к определенным классам и функциям проектов, процессов и компонентов программных комплексов. При этом должна сохраняться концептуальная целостность применяемой совокупности стандартов. Должно обеспечиваться их положительное влияние на процессы и результаты, на качество, надежность и безопасность программных продуктов при реальных ограничениях на использование доступных для их сопровождения ресурсов в жизненном цикле проектов.

Методология программной инженерии и международные стандарты рекомендуют современные методы и средства проектирования и производства, внедрения и применения программных продуктов в составе сложных систем. Организованные коллективы специалистов, применяющие регламентированные стандартами процессы программирования, верификации, тестирования и сопровождения программных комплексов и их компонентов обеспечивают заказчикам стабильные, предсказуемые результаты – программные продукты с требуемым качеством обработки информации. Основной интеллектуальный труд специалистов вкладывается в разработку функциональных алгоритмов и текстов программ, а также в их интеграцию, испытания, документирование на всех этапах проектирования и про-

изводства сложных программных продуктов. Большая сложность информации в таких системах определяет высокие затраты квалифицированного интеллектуального труда специалистов на комплексы программ, для материализации информации в программном продукте.

Методология программной инженерии и стандарты регламентируют также современные процессы управления проектами сложных систем и программных продуктов. Они обеспечивают организацию, освоение и применение апробированных годами, высококачественных процессов проектирования, программирования, верификации, тестирования и сопровождения программных комплексов и их компонентов. Многообразие классов и видов сложных комплексов программ, обусловленное различными функциями и сферами применения систем, определяет формальные трудности, связанные с методами и процедурами доказательства соответствия создаваемых и поставляемых программных продуктов условиям контрактов, требованиям заказчиков и потребителей. По мере расширения применения и увеличения сложности систем выделились области, в которых дефекты, недостаточное качество комплексов программ или данных могут наносить значительный ущерб, намного превышающий положительный эффект от их использования. В таких критических системах (например, управления атомными электростанциями, крупными банками или системами вооружения) недопустимы проявления высоких рисков, нарушений принятых режимов функционирования программных продуктов при любых искажениях исходных данных, сбоях, частичных отказах аппаратуры, ошибках пользователей и других нештатных ситуациях. Подобные риски комплексов программ могут определять безопасность функционирования объектов, предприятий и даже страны. Вследствие этого резко повысилась ответственность специалистов за качество результатов их труда и создаваемых программных продуктов. Это требует непрерывного совершенствования, обучения и повышения квалификации заказчиков, разработчиков и пользователей в области программной инженерии, освоения ими современных методов, процессов и международных стандартов, а также высокой корпоративной культуры коллективов специалистов, обеспечивающих жизненный цикл критических программных продуктов.

Суммируя итоги приведенного выше обзора, мы приходим к выводу, что реализация программного проекта обусловлена пятью взаимосвязанными аспектами, которые условно называют «пятиугольником программной инженерии». Эти аспекты следующие:

1 – жизненный цикл разработки ПО;

2 – язык моделирования ПО;

3 – инструментальные средства программной инженерии;

4 – планирование работ над проектом ПО;

5 – управление процессом разработки и сопровождения ПО.

Более подробно указанные аспекты программной инженерии рассмотрены и детализированы в последующих главах.

Контрольные вопросы и задания

1. Объясните, как программная инженерия и инженерия систем соотносятся друг с другом. Является ли это отношением включения или пересечения? Может быть, эти две концепции вообще не имеют друг к другу никакого отношения?

2. Каковы пять главных аспектов программной инженерии? Можете ли вы сказать, что любой проект программной инженерии обязательно охватывает все эти аспекты?

3. Переведите на русский язык один из разделов SEBOK и составьте реферат по этому материалу

4. Составьте реферат по одному из проектов, описанных в седьмом разделе SEBOK

Глава 4

СВОД ЗНАНИЙ ПО ПРОГРАММНОЙ ИНЖЕНЕРИИ

Как отмечалось в предыдущих главах, программная инженерия весьма обширная область, связанная со многими областями знаний и технологий, и этим определяются требования к знаниям специалистов, работающих в этой области.

В результате многолетней работы большой группы специалистов были собраны и систематизированы знания, необходимые для работы в области программной инженерии. Полученный свод знаний представлен в виде стандарта ISO/IEC TR 19759:2005 SWEBOK (Software Engineering Body of Knowledge) – Руководство по своду знаний по программной инженерии, который определяет и описывает области знаний, необходимых программному инженеру.

Настоящая глава посвящена знакомству с этим стандартом. Как и большинство материалов, связанных с программной инженерией, SWEBOK – весьма объемный документ, содержащий 335 страниц основного текста и множество приложений, напечатанных мелким шрифтом. Поэтому мы ограничимся краткими рефератами всех разделов SWEBOK, содержащими описание Областей Знаний – Knowledge Area (KA), и сопровождающими их таблицами. Далее мы будем использовать аббревиатуру KA без пояснений. Более тесное знакомство с разделами этого англоязычного документа может понадобиться при дальнейшем прохождении магистерского курса.

В соответствии с рекомендациями SWEBOK, важнейшие аспекты деятельности в сфере программной инженерии (Software Engineering, SWE) относятся к следующим 15 областям знаний, из которых первые семь составляют этапы жизненного цикла ПО:

1. Требования к программам.
2. Проектирование ПО.
3. Конструирование ПО.
4. Тестирование ПО.
5. Эксплуатация и сопровождение ПО.
6. Управление конфигурацией ПО.
7. Управление инженерной деятельностью по созданию ПО.
8. Процессы программной инженерии.
9. Модели и методы программной инженерии.
10. Качество ПО.
11. Управление персоналом в программной инженерии.
12. Экономика программной инженерии.
13. Компьютерные основы программной инженерии.

14. Математические основы программной инженерии.
15. Инженерные основы программной инженерии.

В стандарте SWEBOK каждая из перечисленных областей знаний подробно описана.

Область компетенций программной инженерии граничит со следующими дисциплинами:

1. Математика.
2. Информатика.
3. Вычислительная техника.
4. Экономика.
5. Системная инженерия.
6. Управление проектами.
7. Управление качеством.
8. Менеджмент.

Как мы увидим в гл. 6, области компетенций, определенные стандартом SWEBOK, в значительной мере совпадают с требованиями отечественного образовательного стандарта по программной инженерии при подготовке магистров по направлению 09.04.04.

4.1. Краткое содержание SWEBOK V.3

История создания SWEBOK

В 1958 году Джон Туки, всемирно известный статистик, придумал термин Software (имеется в виду, что «soft – мягкая» часть компьютерной науки, связанная с программированием, в отличие от Hardware («hard –жесткая» часть этой науки, которая связана с аппаратными средствами). В отечественной литературе термин Software означает «Программное обеспечение». Термин «программное обеспечение техники» был использована в названии конференции НАТО, состоявшейся в Германии в 1968 году. Общество IEEE Computer Society впервые опубликовало свои материалы по разработке программного обеспечения в 1972 году, а комитет по созданию стандартов разработки программного обеспечения был создан Обществом IEEE Computer Society в 1976 году.

Каждая профессия основана на совокупности знаний, хотя эти знания не всегда описаны в явном виде. В тех случаях, когда отсутствует формальное описание, совокупность знаний основана на «общепризнанных» практиках и может быть представлена различными способами для различных целей. Но во многих случаях руководство по совокупности знаний официально документируется, как правило,

в форме, которая позволяет использовать ее для таких целей, как разработка и аккредитация академических и учебных программ, сертификация специалистов или профессиональное лицензирование. Как правило, профессиональное сообщество или аналогичный орган поддерживает формальное определением совокупности знаний.

В 1990 году началось планирование создания международного стандарта, обеспечивающего общие правила разработки программного обеспечения для машиностроения. Стандарт был завершен в 1995 году с обозначением ISO/IEC 12207 и назван «Процессы жизненного цикла программного обеспечения». Опубликована версия IEEE 12207 в 1996 году явилась основой для свода знаний, представленных в SWEBOK 2004. Текущая версия стандарта 12207 обозначается как ISO/IEC 12207: 2008 и представляет собой основу для рассматриваемой версии SWEBOK V3. Документ частично исполняет обязательство Общества по содействию продвижению как теории, так и практики профессии разработчика программного обеспечения.

В создании данного Руководства приняло участие свыше 100 специалистов из разных стран. Документ представили Дик Фэрли, председатель Комитета по программному обеспечению и системам Компьютерного общества IEEE и Дон Шафер, вице-президент Совета по профессиональной деятельности Компьютерного общества IEEE.

Отметим, что данное Руководство не содержит всего объема знаний по разработке программного обеспечения, а скорее служит справочником по знаниям, накопленным в этой области более чем за четыре десятилетия. Несмотря на то, что технология знаний в области разработки программного обеспечения постоянно развивается, данное Руководство по-прежнему служит основой и ориентиром при подготовке специалистов по программной инженерии.

Важно понимать, что программная инженерия является развивающейся дисциплиной. Более того, данная дисциплина не касается вопросов конкретизации применения тех или иных языков программирования, архитектурных решений или, тем более, рекомендаций, касающихся более или менее распространенных, или развивающихся с той или иной степенью активности технологий (например, web-служб). Руководство к своду знаний, каковым является SWEBOK, включает базовое определение и описание областей знаний (например, конфигурационное управление – configuration management) и, безусловно, является недостаточным для охвата всех вопросов, относящихся к вопросам создания программного обеспечения, но, в то же время необходимым для их понимания. Поэтому руководство

актуально и представляет собой ценное пособие для профессионального разработчика программного обеспечения.

В настоящем учебном пособии нет возможности поместить весь перевод SWEBOK из-за его объема (335 страниц, 6.8 Гбайт). Поэтому мы ограничиваемся знакомством с предисловиями ко всем 15 главам документа, где кратко изложено их содержание, и приводим таблицы, показывающие структуру областей знаний в каждом разделе. Перевод части разделов с комментариями содержится в работе С. Орлика.

В процессе освоения магистерского курса «Методология программной инженерии» перевод отдельных глав этого документа может послужить студентам хорошей школой знакомства с содержанием знаний по программной инженерии, а также практикой по переводу специальной англоязычной литературы.

4.2. Краткое содержание разделов SWEBOK V3

4.2.1. Управление требованиями

Область знаний о требованиях к программному обеспечению (табл. 4.1) связана с выявлением, анализом, спецификацией и проверкой требований к программному обеспечению, а также с управлением требованиями на протяжении всего жизненного цикла программного продукта. Широко признано среди исследователей и отраслевых практиков, что проекты программного обеспечения критически уязвимы, когда требования, связанные с деятельностью, плохо выполняются. Программные требования выражают потребности и ограничения на программный продукт, которые способствуют решению какой-либо реальной проблемы.

Термин «инженерия требований» широко используется в этой области для обозначения систематической обработки требований. Для обозначения специалиста по требованиям будет использоваться термин «инженер-программист» или, в некоторых конкретных случаях, «специалист по требованиям», там, где рассматриваемая роль обычно выполняется отдельным лицом, кроме инженера-программиста. Это не означает, однако, что инженер-программист не может выполнять эту функцию. Тема «Процесс требований», предназначена для обеспечения высокоуровневого обзора требований путем определения ресурсов и ограничений, в которых работает этот процесс, и которые действуют для его настройки. Альтернативное построение может использовать структуру на основе продукта (системные требования,

требования к программному обеспечению, прототипы, варианты использования и т. д.).

Таблица 4.1

Программные требования

№	Область знаний	Компоненты области знаний
1	Основы требований	Определение требований Требование к продукту и процессу Функциональные и нефункциональные требования Независимые свойства Количественные требования Системные и программные требования
2	Процесс работы с требованиями	Модели процессов Участники процессов Управление и поддержка процессов Качество и улучшение процессов
3	Извлечение требований	Источники требований Техники сбора требований
4	Анализ требований	Классификация требований Концептуальное моделирование Архитектурное проектирование и распределение требований Обсуждение требований
5	Спецификация требований	Документ определения системы Спецификация системных требований Спецификация программных требований
6	Утверждение требований	Сбор требований Прототипирование Утверждение модели Приемочные тесты
7	Практические соображения	Итерационная природа процесса работ с требованиями Управление изменениями Атрибуты требований Трассировка требований Измерение требований

Разбивка на основе процессного подхода отражает тот факт, что процесс требований, для его успеха, должен рассматриваться как непрерывный процесс, включающий сложные, взаимосвязанные действия (как последовательные, так и параллельные), а не как дискретное одноразовое действие в начале проекта разработки программного обеспечения. КА требований к программному обеспечению тесно связаны с разработкой программного обеспечения, его тестировани-

ем, обслуживанием управлением конфигурацией, процессом разработки, моделями, методами разработки и качеством программного обеспечения.

Программные требования – *Software Requirements* – свойства, которые должны быть надлежащим образом представлены для решения конкретных практических задач. Данная область знаний касается вопросов извлечения (сбора), анализа, специфицирования и утверждения требований.

Опыт индустрии информационных технологий однозначно показывает, что вопросы, связанные с управлением требованиями, оказывают критически-важное влияние на программные проекты, в определенной степени – на сам факт возможности успешного завершения проектов. Только систематичная работа с требованиями позволяет корректным образом обеспечить моделирование задач реального мира и формулирование необходимых приемочных тестов для того, чтобы убедиться в соответствии создаваемых программных систем критериям, заданным реальными практическими потребностями.

В то же время, возможен, и часто используется на практике и в различных методологиях разработки ПО, альтернативный подход, базирующийся на определении групп требований *к продукту*.

4.2.2. Проектирование программного обеспечения

Проектирование ПО определяется как «процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или компонента» и как «результат этого процесса». Проектирование ПО (табл. 4.2) – это этап жизненного цикла разработки программного обеспечения, в котором анализируются требования к программному обеспечению, чтобы составить описание внутренней структуры ПО, которая послужит основой для его построения. Результат проектирования ПО описывает архитектуру программного обеспечения, то есть структуру программного обеспечения и его объединение в компоненты, а также интерфейсы между этими компонентами. Он также должен описывать компоненты на уровне детализации, что позволяет их создавать. Проектирование играет важную роль при создании программного обеспечения: во время проектирования программного обеспечения разработчики ПО создают различные модели, которые являются своего рода планом решения, которое будет реализовано.

Таблица 4.2

Проектирование программного обеспечения

№	Область знаний	Компоненты области знаний
1	Основы проектирования	Общие концепции проектирования Контекст программного дизайна Процесс проектирования Техники применения
2	Ключевые вопросы проектирования	Параллелизм в проектировании Контроль и обработка событий Ошибки, обработка исключений и защищенность от сбоев Взаимодействие и представление Сохраняемость данных
3	Структура и архитектура ПО	Архитектурные структуры и точки зрения Архитектурные стили (макро-архитектурные шаблоны) Шаблоны проектирования (микро-архитектурные шаблоны) Семейство программ и фреймворков
4	Анализ качества и оценка программного дизайна	Атрибуты качества Анализ качества и техники оценки Измерения
5	Нотации проектирования ПО	Структурные описания (статическое представление) Поведенческие описания (динамическое представление)
6	Стратегии и методы проектирования	Общие стратегии Функционально ориентированный (структурный) дизайн Проектирование на основе структур данных Компонентное проектирование Другие методы

Мы можем анализировать и оценивать эти модели, чтобы определить, позволят ли они выполнять различные требования. Мы также можем исследовать и оценивать альтернативные решения и компромиссы.

Наконец, мы можем использовать полученные модели для планирования последующих действий по разработке, таких как проверка системы и валидация, в дополнение к использованию их в качестве исходных данных и отправной точки для построения и тестирования.

В стандартном списке процессов жизненного цикла ПО, например, в стандарте ISO/IEC/IEEE Std. 12207, «Процессы жизненного цикла программного обеспечения», разработка программного обеспе-

чения состоит из двух видов деятельности, которые соответствуют анализу требований к программному обеспечению и разработке программного обеспечения:

1. Архитектурный дизайн программного обеспечения (иногда называемый высокоуровневым дизайном): разрабатывает структуру верхнего уровня и организацию программного обеспечения и идентифицирует различные компоненты.

2. Детальный дизайн программного обеспечения: подробно описывается каждый компонент ПО для облегчения его построения. В области знаний Software Design не обсуждается каждая тема, включающая слово «дизайн». Темы, обсуждаемые в этой KA, касаются главным образом D-дизайна (дизайна декомпозиции), цель которого – сопоставление программного обеспечения с компонентами. Однако из-за его важности в области архитектуры программного обеспечения мы также рассмотрим FP-дизайн (дизайн семейных шаблонов), целью которого является установление эксплуатационной общности в семействе программных продуктов. Эта KA не рассматривает I-дизайн (дизайн изобретения), который обычно выполняется во время процесса требований к программному обеспечению с целью концептуализации и определения программного обеспечения для удовлетворения обнаруженных потребностей и требований, поскольку эта тема считается частью процесса требований (см. KA «Требования к программному обеспечению»). Эта KA дизайна ПО связана конкретно с требованиями к программному обеспечению: проектирование, управление программным обеспечением, модели и методы разработки программного обеспечения, качество программного обеспечения и вычислительные основы.

4.2.3. Конструирование программного обеспечения

Термин «конструкция программного обеспечения» относится к детальному созданию рабочего программного обеспечения посредством комбинации кодирования, проверки, модульного тестирования, тестирования интеграции и отладки. Область знаний KA Software Construction (табл. 4.3) связана со всеми другими KA, но она наиболее тесно связана с Software Design и Software Testing, KA, потому что процесс создания программного обеспечения включает в себя трудоемкую разработку и тестирование ПО. Этот процесс использует результаты дизайна и передает свои результаты для тестирования; (слова «дизайн» и «тестирование» в этом случае относится к видам

деятельности, а не к КА). Границы между дизайном, конструированием и тестированием (если таковые имеются) будут варьироваться в зависимости от процессов жизненного цикла ПО, которые используются в проекте. Хотя некоторые детализированные конструкции могут быть созданы до начала конструирования, во время этапа конструирования выполняются многие проектные работы. Таким образом, Software Construction КА тесно связан с Software КА. На протяжении всей разработки инженеры-разработчики программного обеспечения тестируют свои разработки и интегрируют их в систему.

Таблица 4.3

Конструирование программного обеспечения

№	Область знаний	Компоненты области знаний
1	Основы конструирования	Минимизация сложности Ожидание изменений Конструирование с возможностью проверки Стандарты в конструировании
2	Управление конструированием	Модели конструирования Планирование конструирования Изменения в конструировании
3	Практические соображения	Проектирование в конструировании Языки конструирования Кодирование Тестирование Повторное использование Качество Интеграция

Таким образом, КА программного обеспечения тесно связано с КА тестирования ПО. При конструировании программного обеспечения обычно производится наибольшее количество элементов конфигурации, которыми необходимо управлять в программном проекте (исходные файлы, документация, тестовые примеры и т. д.). Таким образом, КА программного обеспечения также тесно связано с КА Software Configuration Management. Хотя качество программного обеспечения важно во всех КА, код является конечной версией программного проекта, и, таким образом, КА качества программного обеспечения тесно связано с КА Software. Поскольку разработка программного обеспечения требует знания алгоритмов и методов кодирования, она тесно связана с КА Computing Foundations, посвященной основам компьютерной науки, которые поддерживают проектирова-

ние и построение программных продуктов. Она также связана с управлением проектами, поскольку управление конструированием может представлять значительные проблемы.

4.2.4. Тестирование программного обеспечения

Тестирование программного обеспечения состоит из *динамической* проверки того факта, что программа обеспечивает *ожидаемое поведение* в *конечном* наборе тестовых примеров, подходящим образом *выбранном* из обычно бесконечного набора (табл. 4.4). В приведенном выше определении выделенные курсивом слова соответствуют ключевым вопросам при описании области знаний тестирования программного обеспечения.

Таблица 4.4

Тестирование программного обеспечения

№	Область знаний	Компоненты области знаний
1	Основы тестирования	Терминология тестирования Ключевые вопросы Связь тестирования с другой деятельностью
2	Уровни тестирования	Над чем проводятся тесты Цели тестирования
3	Техники тестирования	Техники, базирующиеся на интуиции и опыте Техники, базирующиеся на спецификации Техники, ориентированные на дефекты Техники, базирующиеся на условия использования Техники, базирующиеся на природе приложения Выбор и комбинация различных техник
4	Измерение результатов тестирования	Оценка программы в процессе тестирования Оценка выполненных тестов
5	Процесс тестирования	Практические соображения Тестовые работы

Динамическая: этот термин означает, что тестирование всегда подразумевает выполнение программы на выбранных наборах входов. Если быть точным, входное значение не всегда достаточно для проведения теста, поскольку сложная недетерминированная система может реагировать на один и тот же ввод с различным поведением в зависимости от состояния системы. Однако в этом КА термин «вход» означает соглашение, что его значение также включает заданное входное состояние в тех случаях, для которых это важно. Статические методы

также рассматриваются в КА качества программного обеспечения. Они дополняют динамическое тестирование. Стоит отметить, что терминология различается среди разных сообществ, а некоторые используют термин «тестирование» также применительно к статическим методам.

Конечный: даже в простых программах, так много теоретически возможных примеров, что для исчерпывающего тестирования могут потребоваться месяцы или даже годы. Вот почему на практике полный набор тестов обычно можно считать бесконечным, а тестирование проводится по подмножеству всех возможных тестов, которое определяется критериями риска и приоритетности. Тестирование всегда подразумевает компромисс между ограниченными ресурсами и графиками, с одной стороны, и неограниченными требованиями к тестированию – с другой.

Выбор: многие предлагаемые методы тестирования существенно различаются в том, как выбирается тестовый набор, и разработчики программного обеспечения должны знать, что различные критерии отбора могут давать значительно различающиеся степени эффективности. Определение наиболее подходящего критерия выбора в данных условиях является сложной проблемой; на практике применяются методы анализа рисков и экспертиза программного обеспечения.

Ожидаемое поведение: должно быть возможно, хотя и не всегда легко, решить: приемлемы ли наблюдаемые результаты тестирования программы или нет, в противном случае усилия по тестированию бесполезны. Наблюдаемое поведение может быть проверено на предмет потребностей пользователей (обычно называемое тестированием для проверки), в отношении спецификации (тестирование для проверки спецификации) или, возможно, неожиданного поведения из-за неявных требований или ожиданий (см. КА Приемочные тесты в требованиях к программному обеспечению). В последние годы созрел конструктивный взгляд на тестирование программного обеспечения. Тестирование больше не рассматривается как активность, которая начинается только после завершения фазы кодирования с ограниченной целью обнаружения сбоев. Тестирование программного обеспечения должно выполняться повсеместно на протяжении всего цикла разработки и обслуживания. В самом деле, планирование тестирования ПО должно начинаться с ранних этапов процесса определения требований к программному обеспечению, а планы и процедуры тестирования должны систематически и непрерывно развиваться – и, возможно, уточняться – по мере развития программного обеспечения. Эти мero-

приятия по планированию тестирования и непосредственно тестированию оказываются полезными для разработчиков ПО, они помогают выявить потенциальные недостатки, такие как ошибки в проектах, противоречия или упущения, двусмысленности в документации. Для многих организаций требования к качеству программного обеспечения является одним из способов профилактики: очевидно, что гораздо лучше предотвратить проблемы, чем исправить их. Таким образом, тестирование можно рассматривать как средство предоставления информации о функциональности и качественных атрибутов ПО, а также для выявления сбоев в тех случаях, когда предотвращение ошибок не было эффективным. Очевидно, что программное обеспечение все еще может содержать ошибки, даже после завершения обширной тестовой деятельности. Ошибки ПО, возникшие после поставки заказчику, устраняются путем корректирующего обслуживания. Темы обслуживания программного обеспечения рассматриваются в KA Software Maintenance. В область знаний качества программного обеспечения (см. «Методы управления качеством программного обеспечения»). Методы управления качеством программного обеспечения разделяются на статические методы (без выполнения кода) и динамические (с выполнением кода). Обе категории полезны. Рассматриваемая KA фокусируется на динамических методах.

Тестирование программного обеспечения также связано с разработкой программного обеспечения (см. KA «Строительное тестирование в программной конструкции»). В частности, тестирование модулей и их интеграции тесно связано с построением программного обеспечения, если оно не является его частью.

4.2.5. Эксплуатация и сопровождение программного обеспечения

Усилия по разработке программного обеспечения завершаются поставкой программного продукта, который удовлетворяет требованиям пользователя. Соответственно, программный продукт должен меняться или развиваться. После ввода в эксплуатацию обнаруживаются дефекты, происходит изменение рабочей среды и появление новых пользовательских требований. Фаза обслуживания жизненного цикла начинается после гарантийного срока или в процессе поддержки после внедрения, но операции по техническому обслуживанию происходят намного раньше. Обслуживание программного обеспечения является неотъемлемой частью жизненного цикла программного

обеспечения. Однако оно не получило такой же степени внимания, как другие фазы ЖЦ (табл. 4.5). Исторически сложилось так, что в большинстве организаций разработка программного обеспечения имеет гораздо более высокий статус, чем его обслуживание. Это положение сейчас меняется, так как организации стремятся максимально уменьшить свои инвестиции в разработку нового ПО, стремясь сохранить при этом имеющееся программное обеспечение как можно дольше.

Таблица 4.5

Сопровождение программного обеспечения

№	Область знаний	Компоненты области знаний
1	Основы сопровождения	Определения и терминология Природа сопровождения Потребность в сопровождении Приоритет стоимости сопровождения Эволюция программного обеспечения Категории сопровождения
2	Ключевые вопросы сопровождения	Технические вопросы Управленческие вопросы Оценка стоимости сопровождения Измерения в сопровождении
3	Процесс сопровождения	Процессы сопровождения Работы по сопровождению
4	Техники сопровождения	Прикладные программные системы Реинжиниринг Обратный инжиниринг

Парадигма с открытым исходным кодом привлекла дополнительное внимание к проблеме сохранения программных артефактов, разработанных другими. В данном Руководстве сопровождение ПО определяется как совокупность действий, необходимых для обеспечения рентабельной поддержки программного обеспечения (рис. 4.1). Мероприятия выполняются на предшествующей стадии, а также после доставки. Предпусковые мероприятия включают планирование операций по доставке, оценку ремонтопригодности и логистику для перехода к эксплуатации. Деятельность по доставке включает в себя модификацию программного обеспечения, обучение, а также работу или взаимодействие со справочной службой. КА области знаний о поддержке программного обеспечения связана со всеми другими аспектами разработки ПО. Поэтому рассматриваемое описание КА

связано со всеми другими КА руководства по разработке программного обеспечения.

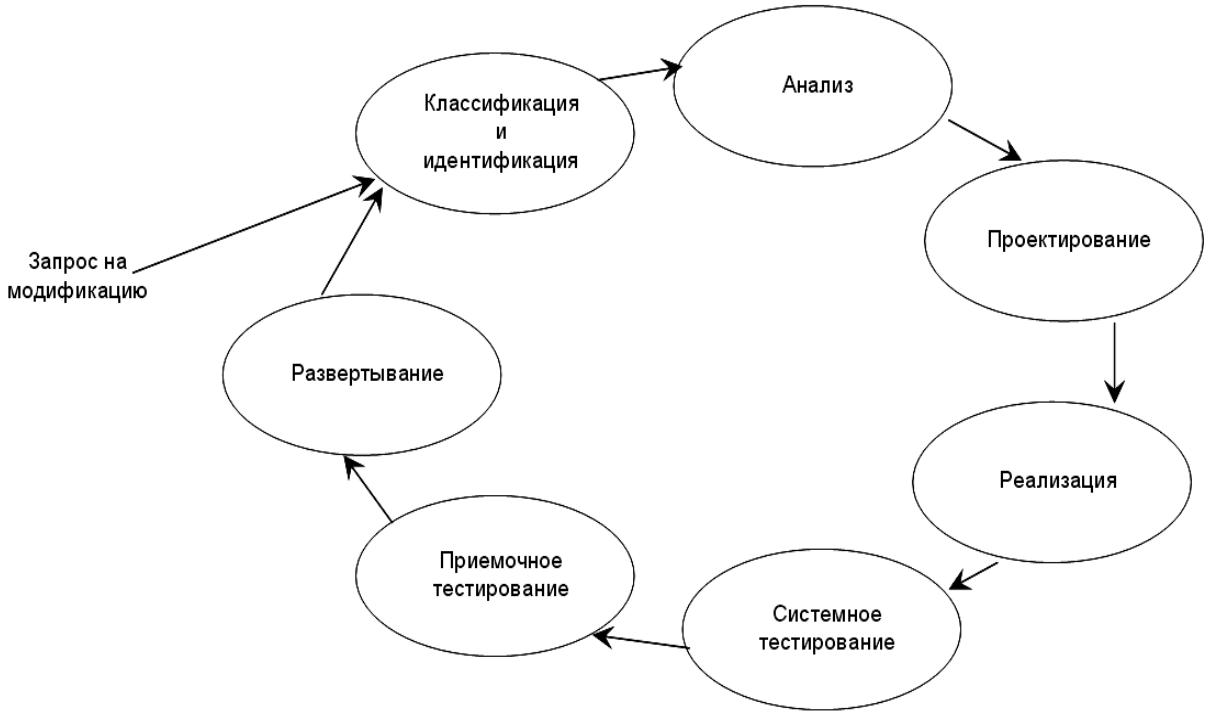


Рис. 4.1. Работы по сопровождению программного обеспечения

4.2.6. Управление конфигурацией программного обеспечения

Как было сказано выше, система может быть определена как комбинация взаимодействующих элементов, организованных для достижения одного или нескольких заявленных целей. Конфигурация системы – это функциональные и физические характеристики аппаратного или программного обеспечения, изложенные в технической документации или достигнутые в продукте; его также можно рассматривать как набор конкретных версий аппаратных средств, прошивки или программных элементов, объединенных в соответствии с конкретными процедурами сборки для выполнения определенной цели. Таким образом, управление конфигурацией (CM, Configuration Management) – это дисциплина по идентификации конфигурации системы в разные моменты времени с целью систематического управления изменениями конфигурации и поддержания целостности и отслеживания конфигурации в течение всего жизненного цикла системы. CM формально определяется как дисциплина, применяющая

техническое и административное руководство и надзор, для следующих целей (табл. 4.6):

– идентифицировать и документировать функциональные и физические характеристики элемента конфигурации;

– контролировать изменения этих характеристик;

– регистрировать и обрабатывать изменения отчета и статус реализации;

– проверять соответствие указанным требованиям.

Таблица 4.6

Конфигурационное управление

№	Область знаний	Компоненты области знаний
1	Управление SCM-процессом	Организационный контекст SCM Ограничения и правила SCM Планирование в SCM План конфигурационного управления Контроль выполнения SCM процессов
2	Идентификация программных конфигураций	Идентификация элементов, требующих контроля Программная библиотека
3	Контроль программных конфигураций	Предложение, оценка и утверждение изменений Реализация изменений Отклонение и отказ от изменений
4	Учет статуса конфигурации	Информация о статусе конфигурации Отчет по статусу конфигурации
5	Аудит конфигурации	Функциональный аудит Физический аудит Внутренний аудит базовых линий
6	Управление выпуском и поставкой	Сборка программного обеспечения Управление выпуском программного обеспечения

Управление конфигурацией программного обеспечения (Software configuration management – SCM) – это процесс жизненного цикла поддержки программного обеспечения, который обеспечивает управление проектами, разработку и обслуживание, деятельность по обеспечению качества, а также обслуживание клиентов и пользователей конечного продукта. Концепции управления конфигурацией применяются ко всем элементам, которые необходимо контролировать, хотя есть некоторые различия в реализации между аппаратным СМ и программным СМ. SCM тесно связан с деятельностью по обеспечению качества программного обеспечения (Software Quality Area –

SQA). Как определено в KA о качестве программного обеспечения, процессы SQA обеспечивают уверенность в том, что программные продукты и процессы в жизненном цикле проекта соответствуют указанным требованиям, а это достигается путем планирования, принятия и выполнения ряда мероприятий, для обеспечения достаточной уверенности в качестве программного обеспечения. Деятельность SCM помогает в достижении этих целей SQA. В некоторых контекстах проекта конкретные требования SQA предписывают определенные действия SCM.

Деятельность SCM – это управление и планирование процесса SCM, идентификация конфигурации программного обеспечения, контроль конфигурации программного обеспечения, учет состояния конфигурации программного обеспечения, аудит конфигурации программного обеспечения и управление выпуском программного обеспечения. KA по управлению конфигурацией ПО относится ко всем другим KA, поскольку объектом управления конфигурацией является артефакт, созданный и используемый в процессе разработки программного обеспечения.

4.2.7. Менеджмент в программной инженерии

Область знаний (KA) по менеджменту программного обеспечения может быть определена как применение управленческой деятельности – планирование, координация, измерение, мониторинг, контроль и отчетность – для обеспечения эффективного использования программных продуктов и программных услуг в интересах заинтересованных сторон. Связанная с ним дисциплина управления является важным элементом всех областей знаний, но, конечно, она более уместна для этой KA, чем для других KA (табл. 4.7). Измерение также является важным аспектом всех KA; тема измерительных программ представлена в этом KA. В некотором смысле должно быть возможным управлять проектом разработки программного обеспечения таким же образом, как управляются другие сложные процессы. Однако существуют специфические аспекты программных проектов и жизненного цикла программного обеспечения, которые усложняют эффективное управление, в том числе следующие:

1. Клиенты часто не знают, что необходимо или что возможно.

2. Клиентам часто не хватает понимания сложностей, присущих разработке программного обеспечения, особенно в отношении воздействия изменяющихся требований.

3. Вероятно, что более глубокое понимание и изменение условий вызовут новые или измененные требования к программному обеспечению.

4. В результате изменения требований программное обеспечение часто создается с использованием итеративного процесса, а не как последовательность закрытых задач.

5. Программное обеспечение обязательно включает в себя творчество и дисциплину. Поддержание надлежащего баланса между ними иногда затруднено.

6. Степень новизны и сложности часто высока.

7. Часто происходят изменения базовой технологии.

Таблица 4.7

Менеджмент в программной инженерии

№	Область знаний	Компоненты области знаний
1	Инициализация и обзор	Определение переговоров по требованиям Анализ осуществимости Процесс обзора и пересмотра требований
2	Планирование проекта ПО	Процесс планирования Определение системы, подлежащей разработке Оценка усилий, графика и стоимости разработки Наличие ресурсов Управление рисками Управление качеством Управление планированием
3	Принятие программного проекта	Включение в планы Получение ПО и заключение контракта с менеджментом Включение процесса менеджмента Процесс мониторинга Процесс управления Процесс оперативного информирования
4	Обзор и оценка	Определение удовлетворительности требований Переписывание и развитие сценария
5	Завершение проекта	Определение процесса завершения Действия по завершения
6	Измерения в программной инженерии	Установление и поддержка обязательств по измерениям План процессов измерений Выполнение процесса измерений
7	Инструменты измерений в программной инженерии	

Деятельность по управлению программным обеспечением осуществляется на трех уровнях: организационное и инфраструктурное управление, управление проектами и управление программой измерений. Последние два подробно описаны в этом КА. Однако это не должно уменьшать важность проблем управления организационной структурой и инфраструктурой. В целом принято решение о том, что менеджеры по организации программного обеспечения должны быть знакомы с управлением проектами и обладать знаниями в области программного обеспечения, описанными в этом КА. Они также должны знать о целевых доменах. Также полезно, если руководители сложных проектов и программ, в которых программное обеспечение является компонентом системной архитектуры, знают о различиях, которые внедряются программными процессами в управление проектами и измерение проекта.

Другие аспекты организационного управления оказывают влияние на разработку программного обеспечения (например, организационные политики и процедуры, которые обеспечивают структуру, в которой осуществляются проекты по разработке программного обеспечения). Возможно, эти политики и процедуры должны быть скорректированы с учетом требований для эффективной разработки и обслуживания программного обеспечения. Кроме того, для эффективного управления разработкой программного обеспечения на организационном уровне может потребоваться целый ряд политик, специфичных для разработки программного обеспечения. Например, политики обычно необходимы для создания конкретных общесистемных процессов или процедур для задач разработки программного обеспечения, таких как проектирование программного обеспечения, разработка программного обеспечения, оценка, мониторинг и отчетность. Такая политика важна для эффективного долгосрочного управления проектами в области разработки программного обеспечения во всей организации (например, создание последовательной основы для анализа эффективности выполнения предыдущих проектов и внедрения улучшений). Еще одним важным аспектом организационного управления является политика и процедуры управления персоналом для найма, обучения и наставничества для развития карьеры не только на уровне проекта, но и для долгосрочного успеха организации. Персонал по разработке программного обеспечения может выполнять уникальные задачи по обучению или управлению персоналом (например, поддержание персонала при изменении курса валют, когда базовая технология подвергается быстрым и непрерывным изменениям).

Управление связью также часто упоминается как упущенный, но важный аспект производительности отдельных лиц в области, где необходимо точное понимание потребностей пользователей, требований к программному обеспечению и программных продуктов. Кроме того, желательно управление портфелем, которое обеспечивает общее представление не только о программном обеспечении, которое в настоящее время разрабатывается в различных проектах и программах (комплексные проекты), но и о программном обеспечении, которое планируется и в настоящее время используется в организации. Кроме того, повторное использование программного обеспечения является ключевым фактором поддержания и повышения производительности и конкурентоспособности. Эффективное повторное использование требует стратегического видения, которое отражает преимущества и недостатки повторного использования. В дополнение к пониманию аспектов управления, которые однозначно влияют на проекты программного обеспечения, разработчики программного обеспечения должны иметь некоторые знания об общих аспектах управления, которые обсуждаются в этой КА (даже в первые несколько лет после окончания обучения). Атрибуты организационной культуры и поведения, а также управление другими функциональными областями предприятия оказывают влияние, хотя и косвенно, на процессы разработки программного обеспечения организации.

4.2.8. Процессы программной инженерии

Инженерный процесс состоит из набора взаимосвязанных видов деятельности, которые преобразуют один или несколько входов в выходы при потреблении ресурсов, необходимых для осуществления преобразования. Многие из процессов традиционных инженерных дисциплин (например, электрических, механических, экономических, химических) связаны с преобразованием энергии и физических объектов из одной формы в другую, как в гидроэлектростанции, которая преобразует потенциальную энергию воды в электрическую энергию или нефтеперерабатывающий завод, использующий химические процессы для превращения сырой нефти в бензин. В КА этой области знаний процессы разработки программного обеспечения связаны с работой, выполняемой разработчиками программного обеспечения для разработки, сопровождения и управления программным обеспечением, таких процессов, как требования, проектирование, разработка, тестирование, управление конфигурацией и другие процессы раз-

работки программного обеспечения. Для удобства чтения, в этом KA «процесс разработки программного обеспечения» будет называться «программным процессом». Кроме того, обратите внимание, что «программный процесс» означает рабочие действия, а не процесс использования внедренного программного обеспечения. Программные процессы определены по ряду причин: для облегчения понимания людьми, общения и координации; для содействия управлению проектами программного обеспечения; для эффективного и качественного измерения качества программных продуктов; для поддержки улучшения процесса; обеспечение основы для автоматизированной поддержки процесса выполнения. SWEBOK KA, тесно связанный с этим процессом разработки ПО, включают в себя управление программным обеспечением, модели и методы разработки программного обеспечения, а также качество программного обеспечения (табл. 4.8). Раздел «Анализ и определение коренных причин», входящий в технические основания KA, также тесно связан с процессом. Software Engineering Management занимается настройкой, адаптацией и внедрением программных процессов для конкретного программного проекта (см. «Планирование процесса в Software Engineering Management KA»).

Таблица 4.8

Процесс программной инженерии

№	Область знаний	Компоненты области знаний
1	Реализация и измерение процесса	Инфраструктура процесса Цикл управления программным процессом Модели реализации и изменения процесса Практические соображения
2	Определение процесса	Модели жизненного цикла Процессы жизненного цикла Нотации определения процесса Адаптация процесса Автоматизация
3	Оценка процесса	Модели оценки процесса Методы оценки процесса
4	Измерение процессов и продуктов	Измерения процессов Измерения продуктов Качество результатов измерений Техника количественной оценки

Модели и методы поддерживают системный подход к разработке и модификации программного обеспечения. Качество программно-

го обеспечения КА связано с процессами планирования, обеспечения и контроля качества проекта и продукта. Результаты измерений в инженерных фондах КА необходимы для оценки и контроля процессов программного обеспечения.

4.2.9. Инструменты и методы программной инженерии

Инструменты и методы разработки программного обеспечения направлены на то, чтобы сделать эту деятельность систематической, повторяемой и, в конечном счете, более ориентированной на успех. Использование моделей обеспечивает подход к решению проблем, нотацию и процедуры построения и анализа моделей. Методы обеспечивают подход к систематической спецификации, разработке, конструированию, тестированию и проверке программного обеспечения конечного продукта и связанных с ним рабочих продуктов. Модели и методы разработки программного обеспечения сильно различаются по масштабам – от решения одной фазы жизненного цикла до полного жизненного цикла программного обеспечения. Акцент в этой области знаний (КА) основан на моделях и методах разработки программного обеспечения, которые охватывают несколько этапов жизненного цикла программного обеспечения, поскольку методы, специфичные для отдельных фаз жизненного цикла, охватываются другими областями знаний КА. Эта глава о моделях и методах разработки программного обеспечения разделена на четыре основные темы:

1. Моделирование: обсуждается общая практика моделирования и дается представление о принципах моделирования; свойствах и видах моделей; синтаксисе моделирования, семантике и прагматике, предпосылках, постусловиях и инвариантах.

2. Типы моделей: кратко обсуждаются модели и агрегация подмоделей и даются некоторые общие характеристики типов моделей, обычно встречающихся в практике разработки программного обеспечения.

3. Анализ моделей: представлены некоторые общие методы анализа, используемые в моделировании, для проверки полноты, последовательности, правильности, прослеживаемости и взаимодействия.

4. Методы разработки программного обеспечения: представлен краткий обзор широко используемых методов разработки программного обеспечения. Обсуждается сравнение эвристических методов, формальных методов, прототипирования и гибких методов. Разбивка тем для моделей и методов разработки программного обеспечения КА показана в табл. 4.9.

Таблица 4.9

Инструменты и методы программного обеспечения

№	Область знаний	Компоненты области знаний
1	Инструменты	Инструменты работы с требованиями Инструменты проектирования Инструменты конструирования Инструменты тестирования Инструменты сопровождения Инструменты конфигурационного управления Инструменты управления инженерной деятельностью Инструменты поддержки процессов Инструменты обеспечения качества Дополнительные аспекты инструментального обеспечения
2	Методы	Эвристические методы Формальные методы Методы прототипирования

4.2.10. Качество программного обеспечения

Что такое качество программного обеспечения и почему так важно, чтобы это понятие было включено во многие области знаний (KA) Руководства SWEBOK?

Одна из причин заключается в том, что термин «качество программного обеспечения» перегружен. Качество программного обеспечения может относиться к желаемым характеристикам программных продуктов в той степени, в которой определенный программный продукт обладает этими характеристиками, а также к процессам, инструментам и методам, используемым для достижения этих характеристик.

На протяжении многих лет авторы и организации определяли термин качество по-разному. Это было «соответствие требованиям», либо «достижение отличных уровней «пригодности для использования». Между тем, IBM придумала термин «рыночное качество», где «клиент является конечным арбитром». Совсем недавно качество программного обеспечения стало определяться как «способность программного продукта удовлетворять заявленным и подразумеваемым потребностям в определенных условиях» и как «степень соответствия программного продукта установленным требованиям»; однако качество зависит от того, насколько эти установленные требования точно отражают потребности, желания и ожидания заинтересованных сто-

рон. Оба определения содержат предпосылку соответствия требованиям. Но ни одно из них не относится к типам требований (например, функциональность, надежность, производительность или любые другие характеристики). Примечательно, однако, что эти определения подчеркивают, что качество зависит от требований. Эти определения также иллюстрируют еще одну причину распространенности термина о качестве программного обеспечения в настоящем Руководстве: частая неоднозначность термина качества программного обеспечения по сравнению с требованиями к качеству программного обеспечения. Требования к качеству программного обеспечения на самом деле являются атрибутами (или ограничениями) функциональных требований (что делает система). Программные требования могут также указывать использование ресурсов, протокол связи или многие другие характеристики.

В нашем изложении КА используется определение качества программного обеспечения в широком смысле на основе приведенных выше определений, и определяются требования к качеству программного обеспечения в виде ограничений функциональных требований (табл. 4.10).

Таблица 4.10

Качество программного обеспечения

№	Область знаний	Компоненты области знаний
1	Основы качества	Культура и этика программной инженерии Значение и стоимость качества Модели и характеристики качества Повышение качества
2	Процессы управления качеством	Подтверждение качества Проверка и аттестация (V&V) Оценка и аудит
3	Практические соображения	Требования к качеству Характеристика дефектов Техника управления качеством Количественная оценка качества

Качество программного обеспечения достигается за счет соответствия всем требованиям независимо от того, какая характеристика указана или как сгруппированы, или названы требования. Качество программного обеспечения также рассматривается во многих КА SWEBOK, потому что это основной параметр оценки усилий по разработке программного обеспечения. Для всех инженерных продуктов

основная цель – обеспечить максимальную ценность для заинтересованных сторон, одновременно балансируя ограничения затрат и графика разработки; это иногда характеризуется как «пригодность для использования». Значение заинтересованности выражается в требованиях.

Для программных продуктов заинтересованные стороны могут определять цену (то, что они платят за продукт), время выполнения (как быстро они получают продукт) и качество программного обеспечения. Эта КА дает определения и обзор практик, инструментов и методов для определения качества программного обеспечения и его оценки во время разработки, обслуживания и развертывания. В приведенных ссылках даются дополнительные сведения.

4.2.11. Профессиональная практика (управление персоналом) в программной инженерии

Область знаний о профессиональной практике КА Software Engineering связана со знаниями, навыками и отношениями, которыми должны обладать разработчики программного обеспечения, чтобы вести разработку программного обеспечения на профессиональной, ответственной и этичной основе. Из-за широкого применения программных продуктов в социальной и личной жизни качество программных продуктов может оказать глубокое влияние на наше личное благополучие и социальную гармонию. Инженеры-разработчики программного обеспечения должны решать уникальные технические проблемы, производить программное обеспечение с известными характеристиками и надежностью. Это требование относится к инженерам-программистам, которые обладают надлежащим набором знаний, навыков, образования и опыта в профессиональной практике. Термин «профессиональная практика» относится к способу оказания услуг, чтобы достичь определенных стандартов или критериев как в процессе выполнения услуги, так и в конечном продукте, являющемся результатом обслуживания. Эти стандарты и критерии могут включать как технические, так и нетехнические аспекты. Понятие профессиональной практики можно рассматривать как более применимое в тех профессиях, которые имеют общепринятую совокупность знаний; кодексы этики и профессионального поведения с наказаниями за нарушения; принятые процессы аккредитации, сертификации и лицензирования; профессиональные общества, чтобы обеспечить и управлять всеми этими компонентами. Прием в эти профессиональные общества часто основывается на предписанной комбинации образования и опы-

та. Инженер-программист поддерживает профессиональную практику, выполняя всю работу в соответствии с общепринятыми практиками, стандартами и руководящими принципами, специально изложенными одним из профессиональных обществ. Например, Ассоциация вычислительной техники (ACM) и IEEE Computer Society (IEEE CS) создали Кодекс этики и профессиональной практики в области программного обеспечения. Также Британское компьютерное общество (БКС), и Международная федерация по обработке информации (ИФИП) установили аналогичные стандарты профессиональной практики. ISO/IEC и IEEE также предоставили международно признанные стандарты разработки программного обеспечения (см. Приложение В настоящего Руководства). IEEE CS создала две международные сертификационные программы (CSDA, CSDP) и соответствующее Руководство для Группы знаний по программному обеспечению (Руководство SWEBOK). Все это элементы, лежащие в основе профессиональной практики разработки программного обеспечения (табл. 4.11).

Таблица 4.11

Профессиональная практика программной инженерии

№	Область знаний	Компоненты области знаний
1	Профессионализм	Аккредитация, сертификация, лицензирование Коды этики и профессионального поведения Сущность и роль профессиональных сообществ Сущность и роль стандартов программной инженерии Экономическая роль программного обеспечения Рабочие контакты Проблемы с законом Документация Анализ компромиссов
2	Коллективное поведение и психология	Динамика поведения рабочих команд/ групп Индивидуальное знание Проблемы со сложностью Взаимодействие со стейкхолдерами Проблемы неопределенности и неоднозначности Проблемы с мультикультурным окружением
3	Навыки коммуникаций	Чтение, понимание, обобщение Письмо Взаимодействие в группах и командах Навыки презентации

4.2.12. Экономика программной инженерии

Экономика разработки программного обеспечения заключается в принятии решений, связанных с разработкой программного обеспечения в бизнес-контексте (табл. 4.12). Успех программного продукта, сервиса и решения зависит от хорошего управления бизнесом. Тем не менее, во многих компаниях и организациях отношения с программным бизнесом к разработке и разработке программного обеспечения остаются неопределенными. Рассматриваемая область знаний (КА) представляет собой обзор по экономике программного обеспечения. Экономика – это изучение стоимости, затрат, ресурсов и их отношений в данном контексте или ситуации. В процессе разработки программного обеспечения мероприятия имеют свои затраты, и в результате программное обеспечение также имеет экономические характеристики. Экономика в области разработки программного обеспечения позволяет систематически изучать атрибуты программных процессов, которые связывают их с экономическими показателями. Эти экономические меры можно взвешивать и анализировать при принятии решений, которые входят в сферу действия программной организации и тех, которые входят в единую сферу деятельности всего производственного или приобретающего бизнеса.

Экономика разработки программного обеспечения связана с согласованием технических решений программного обеспечения с бизнес-целями организации. Во всех типах организаций – будь то «коммерческая», «некоммерческая» или правительственная – это означает устойчивое пребывание в бизнесе. В «коммерческих» организациях это дополнительно относится к достижению ощутимой отдачи от инвестированного капитала – как активов, так и капитала. Эта КА была разработан таким образом, чтобы охватить все типы организаций, не зависимо от их профиля, портфеля продуктов и услуг, ограничений на владение капиталом и налогообложения. Решения типа «Должны ли мы использовать конкретный компонент?» могут выглядеть легко с технической точки зрения, но могут иметь серьезные последствия для деловой жизнеспособности программного проекта и полученного продукта. Часто инженеры задаются вопросом: относятся ли к ним такие проблемы вообще, поскольку они «только инженеры». Экономический анализ и принятие решений являются важными инженерными соображениями, поскольку инженеры способны оценивать решения как технически, так и с точки зрения бизнеса. Содержание этой области знаний является важным для разработчиков программного

обеспечения, даже если они никогда не участвуют в конкретных бизнес-решениях; у них будет хорошо продуманный взгляд на деловые вопросы и роль технических соображений в принятии бизнес-решений.

Таблица 4.12

Экономика программной инженерии

№	Область знаний	Компоненты области знаний
1	Основы экономики программной инженерии	Финансы, учет, контроль, денежные потоки Процессы принятия решений Оценка, инфляция, амортизация Налогообложение Время – деньги Результативность Эффективность Производительность
2	Экономика жизненного цикла	Продукт, проект, программа, портфолио Жизненный цикл продукта, жизненный цикл проекта Предложение, инвестиционные решения, горизонт планирования, цены и ценообразование Управление представлением, управление доходами Решения по перемещению и прекращению
3	Риск и неопределенность	Цели, оценки, планы Техники оценки Источники неопределенности Установление приоритетов Принятие решений по рискам Принятие решений по неопределенностям
4	Методы экономического анализа	Коммерческий анализ Минимально допустимая степень возврата Возврат инвестиций Возврат стоимости рабочей силы Анализ выгоды и затрат Анализ затрат и результатов Определение точки безубыточности Многомерный анализ Оптимизационный анализ
5	Практические соображения	Принцип достаточности Экономия «без трения» Эко-системный подход Оффшоринг и аутсорсинг

Многие инженерные предложения и решения, такие как выбор варианта покупки, имеют глубокие внутренние экономические последствия, которые следует рассматривать в явном виде. Эта КА сначала описывает основы, ключевую терминологию, базовые концепции и общие практики в области разработки программного обеспечения, чтобы указать, как включать принятие решений в разработку программного обеспечения или включать бизнес-перспективу. Затем она предоставляет перспективу жизненного цикла, выделяет управление рисками и неопределенностями и показывает, как используются методы экономического анализа. Некоторые практические соображения завершают описание данной области знаний.

4.2.13. Компьютерные основы программной инженерии

Ниже приводится список областей знаний, относящихся к компьютерным основам программной инженерии.

Объем области знаний о вычислительных средствах включает в себя разработку и операционную среду, в которой программное обеспечение развивается и выполняется. Поскольку никакое программное обеспечение не может существовать в вакууме или работать без компьютера, ядром такой среды является компьютер и его различные компоненты. Знания о компьютере и его основополагающих принципах аппаратного и программного обеспечения служат основой для разработки программного обеспечения. Таким образом, все разработчики программного обеспечения должны хорошо понимать область знаний КА «Основы вычислений». Общепризнано, что разработка программного обеспечения основывается на информатике. Например, «Software Engineering 2004: Руководство по учебным программам для программ бакалавриата в области разработки программного обеспечения» четко гласит: «Одним из особенно важных аспектов является то, что разработка программного обеспечения основывается на компьютерных науках и математике».

Стив Токей написал в своей книге Return on Software: «И компьютерная наука, и разработка программного обеспечения связаны с компьютерами, вычислительной техникой и программным обеспечением. Наука вычисления, как совокупность знаний, лежит в их основе. Разработка программного обеспечения связана с применением компьютеров, вычислительной техники и программного обеспечения в практических целях, в частности, при проектировании, строительстве и эксплуатации эффективных и экономичных программных сис-

тем». Таким образом, в основе разработки программного обеспечения лежит понимание информатики. В то же время как мало кто откажется от роли компьютерных наук в разработке программного обеспечения как в качестве дисциплины, так и в качестве совокупности знаний, нельзя переоценить важность информатики для разработки программного обеспечения; таким образом, этим определяется важность КА по вычислительным основам.

Большинство тем, обсуждаемых в Computing Foundations КА, также являются темами обсуждения на базовых курсах, которые входят в учебные планы компьютерных и магистерских программ. Такие курсы включают программирование, структуры данных, алгоритмы, компьютерную организацию, операционные системы, компиляторы, базы данных, сети, распределенные системы и т. д. Таким образом, может возникнуть соблазн разложить Computing Foundations КА в соответствии с этими часто встречающимися разделами в соответствующих курсах. Тем не менее, чистое разделение по темам имеет серьезные недостатки. Во-первых, не все курсы по информатике связаны или важны для разработки программного обеспечения. Таким образом, некоторые темы, которые в противном случае были бы рассмотрены в ходе курса по информатике, не рассматриваются в этом КА. Например, компьютерная графика, являясь важным курсом в области компьютерной науки, не входит в эту КА. Во-вторых, некоторые темы, обсуждаемые в этом руководстве, не существуют как самостоятельные курсы в программах бакалавриата или выпускников по компьютерным наукам. Следовательно, такие темы не могут быть надлежащим образом охвачены в учебном курсе. Например, абстракция – это тема, включенная в несколько различных курсов по информатике; неясно, какое определение абстракции должна учитываться при ориентации на темы учебных курсов. КА Основы вычислений разделены на семнадцать разных тем. Прямая польза темы для разработчиков программного обеспечения – это критерий, используемый для выбора тем для включения в эту КА. Преимущество такой тематической разбивки дает основание полагать, что Computing Foundations – если ее твердо освоить – следует рассматривать как совокупность логически связанных тем, в основе которых лежит проектирование программного обеспечения в целом и конструирование программного обеспечения в частности. КА Основы вычислений тесно связаны с разработкой программного обеспечения, конструированием программного обеспечения, тестированием программного обес-

печения, обслуживанием программного обеспечения, качеством программного обеспечения и КА по математическим основам.

Компьютерные основы программной инженерии:

1. Техника решения задач.
2. Базовые концепции систем.
3. Устройство и организация компьютеров.
4. Абстракция.
5. Алгоритмы и сложность.
6. Основы программирования.
7. Основы языков программирования.
8. Основы компиляторов.
9. Основы операционных систем.
10. Методы и средства отладки программ.
11. Структуры и представление данных.
12. Основы баз данных и управление данными.
13. Параллельные и распределенные вычисления.
14. Основы сетевых коммуникаций.
15. Разработка и управление безопасностью программ.
16. Гуманитарные основы использования компьютеров.
17. Гуманитарные основы разработки программ.

4.2.14. Математические основы программной инженерии

Профессионалы программного обеспечения имеют дело с программами. Программировать можно только то, что обеспечено хорошо понятой, недвусмысленной логикой. Область знаний (КА) математических основ программной инженерии, приведенная ниже, помогает разработчикам программного обеспечения понять эту логику, которая, в свою очередь, переводится на код языка программирования. Математика, которая является основным направлением в этой КА, сильно отличается от типичной арифметики, где рассматриваются и обсуждаются числа. Логика и рассуждения – суть математики, которую должен знать инженер-программист.

Математика связана с изучением формальных систем. Слово «формальных» связано с точностью, поэтому не может быть какой-либо двусмысленной или ошибочной интерпретации некоторого факта. Математика – это изучение отдельных и всех свойств какого-либо понятия. Эта может относиться к числам, символам, изображениям, звукам, видео – почти ко всему. Короче говоря, не только цифры и

вычисления подчиняются являются объектами изучения математики. Напротив, инженер-программист должен иметь представление об абстракции в разнообразных областях приложений. КА «Математические основы программной инженерии» SWEBOK охватывает основные методы определения набора правил для рассуждений в контексте исследуемой системы. В этой КА определяются и обсуждаются методы, которые позволяют инженеру-программисту рассуждать точным (и, следовательно, математическим) стилем. Язык и методы логики, которые обсуждаются здесь, позволяют описать способ математических доказательства, которые позволяют установить истину, не прибегая к числам. Короче говоря, вы сможете написать программу для задачи, только если она будет следовать некоторой логике. Цель этой КА – помочь вам развить навык идентификации и описания такой логики. Акцент делается на то, чтобы помочь вам понять основные понятия, а не оспаривать ваши арифметические способности.

Математические основы программной инженерии:

1. Множества, отношения, функции
2. Основы логики
3. Техника доказательств
4. Основы вычислений
5. Графы и деревья
6. Теория вероятностей
7. Конечные автоматы
8. Грамматики
9. Численные методы, точность, ошибки
10. Теория чисел
11. Алгебраические структуры

4.2.15. Инженерные основы программной инженерии

IEEE определяет технику как «применение систематического, дисциплинированного, количественного подхода к структурам, машинам, продуктам, системам или процессам». В этой главе описываются некоторые инженерные навыки и методы, которые полезны для инженера-программиста. Список соответствующих областей знаний приводится ниже. Основное внимание уделяется тем из них, которые поддерживают другие КА, одновременно сводя к минимуму дублирование тем, рассмотренных в другом месте этого документа. По мере развития теории и практики разработки программного обеспечения становится все более очевидным, что разработка программного обес-

печения – это инженерная дисциплина, основанная на знаниях и навыках, общих для всех инженерных дисциплин. Эта (KA) – область знаний инженерных основ, которые применяются при разработке программного обеспечения и в других инженерных дисциплинах. Темы этой KA включают эмпирические методы и экспериментальные методы; статистический анализ; измерение; инженерный дизайн; моделирование, прототипирование и моделирование; стандарты; анализ основных причин. Применение этих знаний, в случае необходимости, позволит разработчикам программного обеспечения разрабатывать и поддерживать программное обеспечение более эффективно. А выполнение своих работ успешно и эффективно является целью всех инженеров во всех инженерных дисциплинах.

Инженерные основы программной инженерии:

1. Эмпирические методы и техника эксперимента
2. Статистический анализ
3. Измерения
4. Инженерный дизайн
5. Моделирование, симуляция и прототипирование
6. Стандарты
7. Причинно-следственный анализ

Контрольные вопросы и задания

1. Скачайте из интернета текст SWEBOK v3 и переведите один из разделов руководства (по выбору).
2. Сравните знания по программной инженерии из всех разделов SWEBOK с теми знаниями, которые вы получили при обучении в бакалавриате. Составьте реферат по каждому разделу.

Глава 5

ВОПРОСЫ ПРАКТИЧЕСКОЙ ПРОГРАММНОЙ ИНЖЕНЕРИИ

В данной главе, основанной на материале издания (Практическая программная инженерия на основе учебного примера : пер. с англ. / Л. А. Мацяшек, Б. Л. Лонг. М. : Бином. Лаборатория знаний, 2009), будут рассмотрены следующие вопросы, связанные с практической реализацией программных проектов:

– в чем состоит сущность программной инженерии;

– стадии жизненного цикла и модели, в частности, итеративная и пошаговая разработка;

– языки моделирования ПО, в частности, UML – объектно-ориентированный язык моделирования;

– инструментальные средства программной инженерии для управления проектом, моделирования систем, интегрированного коллективного программирования, управления изменением и конфигурацией проекта;

– планирование работы над проектом и оценка бюджета;

– технологии отслеживания хода выполнения проекта;

– управление людскими ресурсами, привлеченными к проекту;

– управление рисками проекта;

– управление качеством ПО;

– управление изменением и конфигурацией.

5.1. Сущность практической программной инженерии

Содержание практической программной инженерии отражается в следующих шести ключевых тезисах:

1. Система ПО меньше, чем информационная система предприятия.

2. Процесс создания и эксплуатации ПО является частью бизнес-процесса.

3. Программная инженерия отличается от традиционной инженерии.

4. Программная инженерия больше, чем программирование.

5. Программная инженерия напоминает моделирование.

6. Система ПО сложна.

Рассмотрим эти тезисы по порядку.

5.1.1. Система ПО меньше, чем информационная система предприятия

Система ПО просто является частью намного большей информационной системы предприятия. Это означает, что разработка системы ПО является только частью (хотя и фундаментальной) разработки информационной системы предприятия. В свою очередь, информационная система предприятия является компонентом предприятия как целого, а предприятие является частью бизнес-среды.

Информационная система обеспечивает формирование и управление информацией в интересах людей. Часть этой информации генерируется автоматически компьютерными системами. Другая информация вводится людьми вручную. Дело в том, что информационные системы – социальные системы, которые включают и используют ПО и другие компоненты в интересах предприятия. Выделяются следующие компоненты информационной системы: *люди, данные/ информация, процедуры, программное обеспечение, аппаратное обеспечение, линии связи.*

Например, система управления счетами банка состоит из кассиров банка, данных/ информации о клиентах и их счетах, процедур, управляющих изъятием и внесением денег, ПО, способного обработать данные/ информацию, аппаратных средств, на которых может работать ПО (включая банковские автоматы), а также персональных и автоматизированных каналов связи, которые объединяют все эти компоненты в единую систему.

5.1.2. Процесс создания и эксплуатации ПО является частью бизнес-процесса

Процессом можно назвать нечто, где запланированы, организованы, скоординированы и выполнены в определенный период времени некоторые виды деятельности по производству продукта или сервиса. Различие между процессом создания и эксплуатации ПО, с одной стороны, и бизнес-процессом, с другой, определяется и связано с продуктом или сервисом, которые ожидается получить от этих процессов. Результат процесса создания и эксплуатации ПО – ПО. Результат бизнес-процесса – бизнес.

Имеются четкие отношения между ПО и бизнесом. ПО – потенциально главный вкладчик в бизнес-успех. ПО – часть бизнеса предприятия, но не наоборот. Цель функционирования предприятия состоит в том, чтобы сформировать цепочку создания ценностей, которая обеспечивает реализацию бизнес-назначения, задач и целей.

Различие между процессом создания, эксплуатации ПО и бизнес-процессом сродни различию между эффективностью процесса и результативностью, о котором говорилось в первой главе. *Эффективность* (efficiency) означает делать что-то правильно. *Результативность* (effectiveness) означает делать правильную вещь. В организационных терминах результативность подразумевает достижение бизнес-назначения, задач и целей. Все они – то, что нужно получить как результат процесса *стратегического планирования,* проводимого на предприятии. Частью стратегического планирования является *бизнес-моделирование.* Следовательно, целью бизнес-процесса является обеспечение результативности.

В противоположность этому процесс создания и эксплуатации ПО должен обеспечить эффективность. Следовательно, возможна ситуация, когда процесс создания и эксплуатации ПО даст очень *эффективный* программный продукт или сервис, который будет *нерезультативен* для бизнеса. В лучшем случае отсутствие результативности может означать нейтральный результат с точки зрения бизнеса. В худшем случае это может сделать бизнес уязвимым для конкурентов и даже привести к банкротству.

Поэтому ясно, что процесс создания и эксплуатации ПО является характерной частью бизнес-процесса, жизненно важной для успеха предприятия. Чтобы обеспечить результативность наряду с эффективностью, процесс создания и эксплуатации ПО должен быть составной частью бизнес-процесса. В конце концов, решение разрабатывать ли программный продукт или сервис в первую очередь будет результатом стратегического планирования и бизнес – моделирования.

С другой стороны, разработка ПО предназначена для поставки программных продуктов и сервисов, увеличивая для предприятия стоимость бизнеса. Это имеет отношение к трем *уровням управления*, которые бизнес-процессы обслуживают: оперативный, тактический и стратегический.

Помещение разработки ПО в среду бизнес-моделирования означает, что процесс создания и эксплуатации ПО получен из более широкой бизнес-модели, и он старается поддерживать и реализовывать конкретный бизнес-процесс в этой модели. Отсюда следует, что программный продукт/ сервис не может быть только информационным сервисом. Он должен также реализовывать бизнес-операции или содействовать им. Проект информационной системы должен или явно

определить бизнес-процесс, который он обслуживает, или, что лучше, он должен быть частью *системы управления знаниями.* Одним из аспектов такого проекта является координация между автоматическими информационными действиями, ручными действиями и творческими действиями по принятию решения.

Обычно система ПО обслуживает один уровень управления – оперативный, тактический или стратегический.

Оперативный уровень обрабатывает оперативные бизнес-данные и документы, типа заказов и счетов. Это – царство OLTP-систем *(online transaction processing* – оперативная обработка транзакций), сопровождаемых обычной технологией *баз данных (БД).*

Тактический уровень обрабатывает информацию, полученную от анализа данных, типа ежемесячных тенденций в заказах продуктов. Это – царство OLAP-систем (online analytical processing – аналитическая обработка в реальном времени), которые сопровождает технология *хранилищ данных.*

Стратегический уровень обрабатывает организационные знания, типа правил и фактов, обусловливающих высокий уровень выгодной продажи изделий. Это – царство систем знаний, которые сопровождает технология *баз знаний.*

Программные продукты/ сервисы на оперативном уровне обязательны для предприятия. Без них современное предприятие не может функционировать.

Однако одно оперативное ПО не дает предприятию никакого конкурентного преимущества. Ведь конкуренты также работают с подобными системами ПО. Бизнес-ценность ПО увеличивается с более высокими уровнями управления, к которым оно применяется.

Интересно, что программные продукты/ сервисы, которые потенциально обладают наибольшей бизнес-ценностью для предприятия, наиболее трудно автоматизировать. И это понятно. Стратегическое управление – из области организационных *знаний* и *мудрости.* Как отмечено Бенсоном и Стандингом: «Мудрость и знания существуют только в умах людей. Когда люди говорят относительно знания чего-то, типа номера телефона, на самом деле они говорят о данных. Понимание того, как использовать эту часть данных – знание.

Решение не вызывать кого-то на работу в 3.00 утра – пример мудрости». Обработка и передача знаний (не будем упоминать мудрость) являются и будут оставаться, главным образом, социальным явлением, мотивационным и интуитивным, но не техническим и не предсказуемым.

5.1.3. Программная инженерия отличается от традиционной инженерии

Тот факт, что система ПО является компонентом информационной системы, подразумевает, что программная инженерия – лишь часть более широкой дисциплины – системной инженерии, о чем уже было сказано ранее. Следовательно, инженер ПО должен понимать требования всей системы и должен быть компетентен в ее предметной области, чтобы проектировать интерфейсы, которыми ПО должно снабдить внешние устройства системы. Инженер ПО должен также понимать, что получение некоторых данных или обработку информации лучше реализовать с помощью аппаратных средств ЭВМ, чем с помощью ПО, и что некоторую обработку не нужно автоматизировать вообще.

Системная инженерия связана с изучаемыми принципами, которые определяют внутреннюю работу сложных систем. Существует длинная история инженерии систем в традиционных технических дисциплинах, типа проектирования механических или электрических систем. Разработанные принципы инженерии систем формализованы в математических моделях. Модели утверждаются и используются в технических изделиях. Эти изделия материальны по своей природе – мосты, строения, электростанции.

С программными продуктами дело обстоит иначе. ПО нематериально по природе. Классические математические модели применяются к некоторым, но не ко всем аспектам ПО. ПО определено в нечетких терминах – «хороший», «плохой», «приемлемый», «удовлетворяющий требованиям пользователя» и т. д. Подобные качества используются в области обслуживания, где качество связано с нечеткими терминами типа «хорошее обслуживание», «удобство клиента», «компетентность», «знание работы» и т. д. Программная инженерия может заниматься «нечеткими» проблемами, но это не подразумевает, что они должны быть менее строгими или недоказуемыми. Очевидно, что программная инженерия должна использовать различные области математики, типа нечеткой логики или нечетких множеств, обеспечивающие строгость и доказательность.

Программная инженерия не должна быть бедной падчерицей традиционной инженерии. Это совсем другое. Никто в традиционной инженерии не ожидает, что мост, построенный по математическим моделям, разрушится. Точно так же «хорошее» ПО, «удовлетворяющее требованиям пользователя», не должно терпеть неудачу. Однако имеется одно «но» – все это при условии, что тем временем реши-

тельно не изменятся требования и ожидания пользователя или внешние обстоятельства.

Никто не ожидает, что мост будет перемешен на десять метров после того, как он был построен. Точно так же не следует ожидать, что программный продукт успешно выполнит различные задачи после того, как будет создан. Если это то, что нам нужно, тогда ПО создано удачно. ПО только тогда должно стать непригодным или недопустимым, когда бизнес создает новые условия или меняется внешняя среда. Если речное русло переместится на десять метров из-за недавнего наводнения, инженер-строитель не может быть обвинен, и не следует ожидать, что существующий мост можно будет легко переместить, чтобы приспособить к новому руслу.

Все сказанное означает, что разработчик ПО должен быть готов создавать ПО, которое можно приспосабливать к изменениям. Этого требует сама природа ПО. ПО должно иметь возможность сопровождения: понятно, ремонтопригодно и расширяемо. Это то, что отличает ПО от моста и делает программную инженерию отличной от традиционной инженерии.

Каждая система ПО уникальна, и процесс ее создания уникален. В отличие от традиционных технических дисциплин прикладной программный продукт не *производится*, он *реализуется (превращается в реальность)*. Эго – не автомобиль или рефрижератор. Программный продукт должен быть реализован, чтобы приспособить его к окружающей среде. Каждый случай системы ПО уникален: либо она построена на пустом месте, либо переделана из имеющегося в наличии коммерческого пакета программ (COTS – commercial off-the-shelf software – коммерческие коробочные программные продукты). Только системное ПО и инструментальные средства ПО, типа операционных систем (ОС) и текстовых процессоров, были когда-то произведены в широком масштабе. *Прикладное ПО*, реализуется, а не производится.

5.1.4. Программная инженерия больше, чем программирование

Как было отмечено ранее, программная инженерия – область информатики, имеющая дело с созданием систем ПО, которые являются настолько большими или настолько сложными, что создаются коллективом или коллективами инженеров.

Здесь важны два понятия: «коллективы людей» и «сложные системы».

В программировании используется термин «разделение кода» – написание серий инструкций, чтобы заставить компьютер выполнить поставленную задачу. Если задача большая, для программирования может быть использован коллектив программистов, но каждый акт программирования – прежде всего персональная деятельность. Программирование – навык. Учитывая определение и спецификацию задачи, программист применяет свои навыки, чтобы выразить проблему на языке программирования.

Программная инженерия больше, чем программирование. Она обращается к сложным проблемам, которые не могут быть решены, используя одно программирование. Сложные системы должны быть разработаны прежде, чем они будут запрограммированы. Подобно строительной индустрии, над сложной системой должен поработать *архитектор*, прежде чем она будет построена. Она должна быть *разбита на модули* используя обобщение и метод «разделяй и властвуй». Каждый модуль затем должен быть тщательно специфицирован и определены его интерфейсы к другим модулям, прежде чем его отдавать программистам для кодирования.

Программист имеет ограниченное понимание всей системы. Он/ она кодирует одновременно один программный модуль – компонент ПО, который должен быть объединен (инженером ПО) с другими компонентами, чтобы сконфигурировать рабочую систему.

Часто инженеру ПО доступны различные версии одного и того же компонента. Конфигурация ПО выполняется объединением определенных версий различных компонентов. По этой причине можно иметь различные конфигурации одной и той же системы.

Прежде чем система будет разработана, инженер ПО должен разобраться с требованиями к ней. Это означает, что должен быть сделан и определен на некотором языке моделирования анализ требований. Стандартный язык моделирования в современной практике – UML (Unified Modeling Language – унифицированный язык моделирования). И анализ, и синтез моделей выполняются в UML.

Инженер ПО создаст такую UML-модель системы, по которой может быть создан исходный код программы. Программисты могут начинать работу с этого момента, но инженер ПО остается ответственным за циклическое проектирование между проектом и кодом. Циклическое проектирование – итеративный процесс, представляющий как прямое (от проекта к коду), так и обратное (от кода к проекту) проектирование.

Наконец, программная инженерия – работа коллектива. Коллективом нужно управлять. Следовательно, программная инженерия тре-

бует управления проектом и воздействует на него. Эта работа включает планирование, составление бюджета и разработку графика, управление качеством и управление рисками, управление конфигурацией и изменениями.

Резюмируя все, можно сказать, что программная инженерия связана с обеспечением структурного решения системы, с проектированием структурных компонентов, с объединением компонентов в рабочую систему, с прямым и обратным проектированием, с руководством проектом и т. д. Программная инженерия – сложный процесс, в пределах которого программирование является полезным ремеслом.

5.1.5. Программная инженерия напоминает моделирование

Модели – абстракции реального мира. Они являются абстрактными представлениями действительности. Абстракция – мощная технология в программной инженерии. Позволяя концентрироваться на важных аспектах проблемы и игнорировать аспекты, которые являются в настоящее время несущественными, абстракция позволяет систематически справляться со сложностью проблемы. Абстракция применяется также и к программным продуктам и процессу создания ПО. Модель процесса создания ПО является абстрактным представлением этого процесса. В практических терминах модель процесса создания ПО определяет стадии жизненного цикла и то, как они взаимодействуют. Модель программного продукта – это его абстрактное представление. Она определяет дискретный продукт в дискретные стадии жизненного цикла.

Модель процесса создания ПО определяет, какие программные продукты, требуемые для обеспечения стадий жизненного цикла, создавать на различных уровнях абстракции.

Далее приведен список моделей базовых программных продуктов:

Модель требований – сравнительно неформальная модель, которая охватывает требования пользователя и описывает систему в терминах ее бизнес-ценности.

Модель спецификаций – модель, которая определяет требования к более формальному использованию терминов, применяя язык моделирования типа UML.

Структурная модель – модель, которая определяет желаемую структуру системы.

Детальная модель проекта – модель, которая определяет характеристики программного/ аппаратного обеспечения, необходимые для реализации программирования.

Программная модель – конструктивная модель, которая представляет окончательную выполнимую модель ПО.

Каждая из этих моделей программного продукта может быть разделена далее для ее детализации. Например, детальная модель проекта может включать модель пользовательского интерфейса, модель БД, модель программной логики и т. д.

Наконец, подход к программной инженерии, используемый при создании системы, влияет на абстракции моделирования. Два главных подхода – в старом стиле функциональной (процедурной, командной, структурной) разработки и в современном стиле объектно-ориентированной разработки.

Функциональный подход разбивает сложную систему до управляемых единиц, используя прием, известный как функциональная декомпозиция. Для этой цели используется технология, называемая моделированием потока данных. Модель ПО последовательно делится на процессы (при уменьшении уровня абстракции), связанные с потоками данных.

Объектно-ориентированный подход разбивает систему на пакеты/ компоненты классов, связанные различными отношениями. Абстракция может применяться, формируя вложенные структуры, т. е. пакет/ компонент может содержать многие уровни других пакетов/ компонентов. Это книжная квинтэссенция объектно-ориентированной программной инженерии.

5.1.6. Система ПО сложна

В прошлом ПО было монолитно и процедурно по своей природе. Типичная программа прошлого, написанная на КОБОЛе, была единственной сущностью, использующей подпрограммы, вызываемые по мере необходимости. Логика программы была последовательна и предсказуема. Сложность такого ПО была просто следствием его размера.

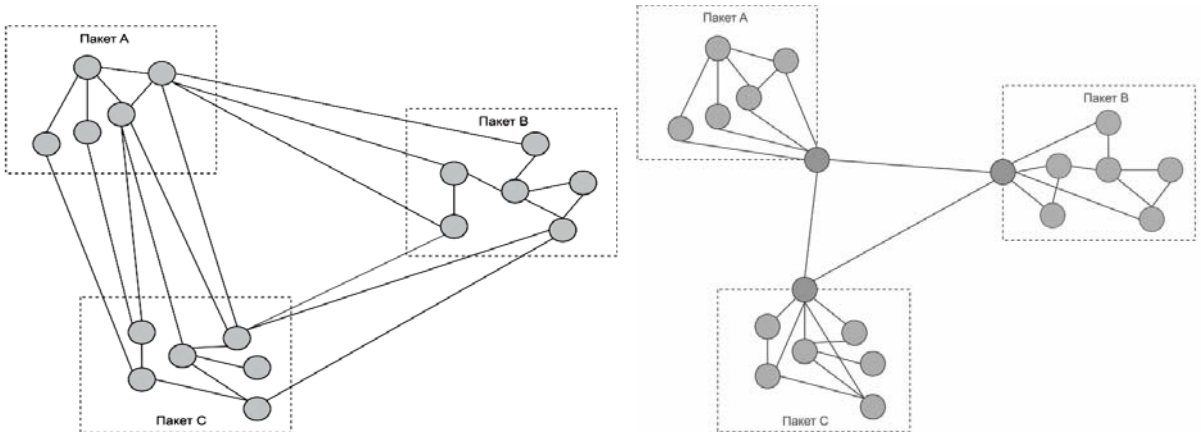
Современное объектно-ориентированное ПО является распределенным (оно может находиться на многих узлах компьютерной сети), и его выполнение случайно и непредсказуемо. Размер современного ПО – сумма размеров его компонентов.

Каждый компонент разработан так, чтобы он был ограниченного управляемого размера. В результате размер не является главным фактором в сложности современного ПО.

Сложность современного ПО заключается в «проводах», то есть в связях и коммуникационных путях между компонентами. Связи между компонентами создают зависимости между распределенными компонентами, которые могут быть сложны для понимания и управления.

Трудность усугубляется тем, что компоненты часто разрабатываются и управляются людьми и коллективами, даже не известными друг другу.

Решение дилеммы находится в замене сетей объектов иерархиями (древовидными структурами) объектов. Все приемлемые сложные системы имеют *иерархическую* форму. На рис. 5.1 показано, как можно уменьшить сложность системы, допуская только единственный канал связи между двумя пакетами.



а *б*

Рис. 5.1. Пример уменьшения сложности системы
путем создания иерархической структуры

Каждый пакет определяет интерфейсный объект (это может быть интерфейс Java-стиля или так называемый доминантный класс), через который осуществляется вся связь с пакетом. Несмотря на добавление трех дополнительных объектов, сложность системы, изображенной на рис. 5.1, *б* явно уменьшена по сравнению с той же самой системой, изображенной на рис. 5.1, *а*.

5.2. Стадии жизненного цикла программного обеспечения

Жизненный цикл ПО – это абстрактное представление процесса создания и эксплуатации ПО. Он определяет для проекта разработки ПО стадии, шаги, действия, методы, инструменты, а также то, что ожидается сдавать. Все это определяет стратегию разработки ПО. Существует ряд полезных моделей жизненного цикла, определяемых стандартами, которые находятся друг с другом в общем согласии по стадиям жизненного цикла, но отличаются важностью отдельных стадий и взаимодействиями между ними.

Стадии жизненного цикла, рассматриваемые в данном пособии, следующие:

1) анализ требований;
2) проектирование системы;
3) реализация;
4) интеграция и внедрение;
5) процесс функционирования и сопровождения.

Рассмотрим их по порядку.

5.2.1. Анализ требований

«Требования пользователя – это утверждения на естественном языке плюс диаграммы, содержащие сведения, какие услуги ожидаются от системы, и ограничения, при которых система должна работать». Анализ требований включает действия по их определению и составлению их списка. В современной практике анализу требований помогает хорошая степень технической строгости, и поэтому эти требования иногда отождествляют с техническими требованиями.

Определение требований, оказывается, одна из самых больших проблем любого жизненного цикла разработки ПО. Пользователям часто неясно, что они требуют от системы. Часто они не знают реальные требования, преувеличивают их, предъявляют требования, которые противоречат требованиям коллег и т. д. Имеется также риск, как и в любом общении между людьми, что истинное значение требования будет неправильно истолковано. Имеется много методов и технологий выявления требований. Они включают:

– интервьюирование пользователей и экспертов в предметной области;

– анкетные опросы пользователей;

– наблюдение, как пользователи выполняют свои задачи;

– изучение существующих документов системы;

– изучение подобных систем ПО, чтобы выяснить состояние в предметной области;

– изучение опытных образцов рабочих моделей для определения и подтверждения требований;

– объединенные совещания разработчиков и клиентов по разработке приложения.

Спецификация требований следует за выявлением требований. В настоящее время UML – стандартный язык моделирования для спецификации требований (так же, как и для проектирования системы). Требования определяются как в графических моделях, так и с помощью текстовых описаний. Поскольку сложную систему нельзя понять с единственной точки зрения, модели снабжаются дополнениями и при необходимости перекрестными точками зрения на систему.

И графические представления, и текст размещаются в хранилище специального CASE-средства. Данное средство облегчает изменения в моделях, если это потребуется. Оно позволяет интегрировать различные модели с перекрывающимися концепциями. CASE-средство позволяет также выполнять преобразования между моделями анализа (где это возможно) и помогает в преобразованиях моделей проектирования.

Анализ требований завершается созданием *технического задания*. Большинство организаций использует некоторые шаблоны для технического задания. Шаблон определяет структуру документа и дает руководящие принципы того, как его писать. Основная часть технического задания содержит модели и описания сервисов и ограничений системы. *Сервисы системы* (то, что система должна делать) часто делятся на функциональные требования и требования к данным. *Ограничения системы* (то, чем система ограничена) включают соображения, связанные с пользовательским интерфейсом, работой, безопасностью, эксплуатационными условиями, политическими и юридическими ограничениями и т. д.

В РФ до сих пор используется документ ГОСТ 34.602–89 Техническое задание на создание автоматизированной системы.

Тестирование абстрактных моделей затруднено, поскольку большую часть времени оно не может быть автоматизировано. *Сквозной контроль* и *инспекции* – вот две популярные и эффективные технологии. Данные технологии схожи. Это предварительные встречи разработчиков и пользователей, на которых «проходятся» по техническому заданию и документам. Обсуждение, которое происходит

на встречах, вероятно, раскроет некоторые проблемы. Сущность этих технологий заключается в том, что в течение встреч идентифицируются проблемы, но их решение не определено, и нет никакого «указующего перста» на людей, потенциально ответственных за эти проблемы. Результаты таких совещаний протоколируются и подписываются всеми их участниками.

5.2.2. Проектирование системы

Проектирование ПО – это описание структуры ПО, которое будет реализовано, данных, которые являются частью системы, интерфейсов между компонентами системы и, иногда, используемых алгоритмов. В информационных системах предприятий структуры данных подразумевают БД. Алгоритмы не всегда полностью описываются во время проектирования, чтобы оставить некоторый уровень свободы выполнения программистам (ведь говоря прямо, проектировщики – это не программисты, и они не в состоянии выбрать умные алгоритмические решения).

Проектирование начинается там, где заканчивается анализ. Столь же истинно и тривиально утверждение, что линия, отделяющая анализ от проектирования, во многих проектах не столь уж и ясна. Теоретически проблема проста. Анализ – моделирование, не ограниченное никакой реализацией (аппаратного/ программного обеспечения). Проектирование – моделирование, которое учитывает платформу, на которой должна быть реализована система.

На практике различие между анализом и проектированием размыто. Этому имеются две главные причины. Во-первых, современные модели жизненного цикла являются *итеративными* и *пошаговыми*. В большинстве таких моделей при разработке в любой момент времени имеются многочисленные разнообразные версии ПО. Некоторые из версий находятся в процессе анализа, другие – в процессе проектирования; некоторые в разработке, другие в производстве и т. д. Во-вторых, и это более важно, для анализа и проектирования используется один и тот же язык моделирования (UML). Переход от анализа к проектированию «по готовности» предпочтительней, чем переход между различными представлениями. Модель анализа уточняется в модели проекта простым дополнением деталей спецификации. Провести линию раздела между анализом и проектированием в таких обстоятельствах очень трудно.

Проектирование, обсужденное выше, более точно называется *детальным проектированием*, т. е. проектированием, которое добав-

ляет детали к моделям анализа. Но имеется другой аспект проектирования системы, а именно структурное проектирование. *Структурное проектирование* связано с определением структуры системы, которая должна быть детально спроектирована и которой следует твердо придерживаться, а также с принципами и образцами внутренних коммуникаций между компонентами.

Структурное проектирование задает «красоту» системы. Главная цель структурного проектирования состоит в том, чтобы получить систему, которая является приемлемой – понятной, ремонтнопригодной и расширяемой. Детальный проект должен соответствовать структурному проекту. Из-за расплывчатой линии раздела между анализом и детальным проектированием некоторые ранние структурные решения, может быть, придется заново выбрать внутри технического задания или даже раньше (но после определения требований).

Тестирование структурного проекта двулико. Во-первых, преимущества структуры, предложенной проектировщикам, должны быть продемонстрированы. Нужно показать, что структура поддерживает сложность ПО, гарантирует возможность сопровождения, упрощает разработку и т. д. Во-вторых, тестирование структурного проекта должно подтвердить, что проект компонентов соответствует принципам и шаблонам принятой структуры.

Тестирование детального проекта также имеет два аспекта. Во-первых, чтобы быть тестируемым, должна быть возможность трассировать детальный проект. Управление трассировкой – целая отрасль программной инженерии, занимающаяся поддержанием связей между продуктами ПО в различных стадиях разработки. В случае детального проекта каждый продукт проекта должен быть связан с требованиями в техническом задании, которое мотивировало производство того продукта. Наличие продукта еще не подразумевает, что это требование обеспечено. Следовательно, второй аспект тестирования проекта использует сквозной контроль и инспекции, чтобы оценить качество изделия проекта.

5.2.3. Реализация

Реализация в большей мере связана с программированием. Но программирование подразумевает не только группу людей, сидящих в общем помещении и кодирующих на некотором языке программирования в соответствии со спецификацией проекта. Программирование предполагает намного более интеллектуальные требования, чем это. Проекты могут быть не доопределены в некоторых областях,

когда они попадают к программистам, особенно в области проектирования алгоритмов. Завершение спецификаций требует дополнительного проектирования прежде, чем можно будет начать кодирование. В этом смысле программист – тоже проектировщик.

Программист – инженер, имеющий дело с компонентами. Сегодняшнее программирование редко выполняется на пустом месте. Большая часть программирования основана на многократном использовании уже созданных компонентов. Это означает, что программист должен иметь знание о компонентах ПО и должен знать, как найти это ПО, чтобы добавить к нему новые закодированные компоненты приложения. Это трудный вопрос.

Программист – инженер, работающий в двух направлениях. Программирование начинается с преобразования проекта в код. Начальный код не должен программироваться вручную. При использовании CASE-средств и IDE-средств (integrated development environments – интегрированные средства разработки) код начинает формироваться (прямое проектирование) из моделей проекта. После того как эта работа будет сделана, произведенный код должен быть скорректирован вручную, чтобы заполнить отсутствующие части (эти «части» существенны и наиболее трудны в программировании). После того как код будет модифицирован программистом, он может обратно воздействовать на модели проекта, корректируя их. Эти технические операции в прямом и обратном направлении называются *циклическим проектированием***.**

Во многих проектах реализация – самая длинная из стадий разработки.

В некоторых моделях жизненного цикла, типа быстрой разработки ПО (раздел 5.3.4), реализация является доминирующей стадией разработки. Реализация – подверженная ошибкам деятельность. Время, потраченное на творческое написание программ, может быть меньше, чем время, потраченное на отладку программы и ее тестирование.

Отладка – это процесс удаления из ПО «блох – bugs» – ошибок в программах. Ошибки в синтаксисе программы и некоторые логические ошибки могут быть определены и исправлены коммерческими средствами отладки. Другие ошибки и дефекты должны быть обнаружены во время тестирования программы. Тестирование может иметь форму просмотра кода (сквозной контроль и инспекции) или может основываться на выполнении программы (наблюдение за поведением программы во время ее выполнения). Управление трассиров-

кой поддерживает возможность использования тестирования, если программы удовлетворяют требованиям пользователя.

Имеются два вида тестирования, основанного па выполнении программы: тестирование на основе технических требований *(«тестирование черного ящика»)* и тестирование на основе кода *(«тестирование белого ящика»)*. Оба вида используют ту же самую стратегию задания программе входных данных и наблюдения, тот ли выходной результат получается, который ожидался. Различие заключается в том, что при тестировании на основе технических требований программе задаются данные без какого-либо учета логики работы программы. Считается, что программа должна вести себя разумно при любых входных данных. В тестировании на основе кода используются такие входные данные, которые позволяют проверить определенные пути выполнения программы – столько путей и настолько разнообразных, насколько это возможно. Поскольку тестирование на основе технических требований и тестирование на основе кода обнаруживают различные виды ошибок и дефектов, нужно использовать их оба.

5.2.4. Интеграция и внедрение

«Целое – больше, чем сумма его частей». Этот афоризм Аристотеля (384–322 гг. до н. э.), называемый в современном системном анализе свойством *эмержентности,* охватывает сущность интеграции системы и ее внедрения. Интеграция собирает приложение из набора компонентов, предварительно созданных и проверенных. Внедрение – передача системы клиентам для использования в производстве.

Интеграция ПО означает переход от «программирования в малом» к «программированию в большом». Информационные системы предприятий – все достаточно большие и сложные, и для них интеграция – существенная стадия в жизненном цикле.

Интеграцию также трудно отделить от тестирования. Фактически, стадия интеграции жизненного цикла часто упоминается и обсуждается под термином тестирования интеграции. При широком использовании итеративных моделей жизненного цикла ПО создается как последовательность быстрых пошаговых реализаций. Каждый шаг – интеграция компонентов, до этого проверенных индивидуально, однако при этом до внедрения саму эту интеграцию системы необходимо сначала проверить.

В значительной степени интеграция определяется структурным проектом системы. В свою очередь, структура системы определяет ее

компоненты и зависимости между ними. Особенно важно, чтобы структурное решение было в виде иерархии или древовидной структуры. Иерархия (древовидная структура) означает устранение любых циклических зависимостей между компонентами. В случае циклических зависимостей тестирование интеграции отдельных шагов создания ПО (конструкций) может оказаться невозможным.

Рассмотрим структуру зависимых компонентов, где компонент С1 использует компонент С2, а С2 использует компонент С3. Предположим, что С1 и С2 уже были реализованы и индивидуально протестированы, а компонент С3 должен быть еще создан. Задача состоит в том, чтобы объединить С1 и С2. Эта задача требует программирования *испытательной заглушки* для С3, то есть части кода, которая моделирует поведение отсутствующего компонента С3. Заглушка обеспечивает среду интеграции для выполнения компонента С2. Обычный путь создания заглушки – сделать ее такой, чтобы она обеспечивала те же зависимости входа/ выхода, что и в окончательном компоненте С3, и формировать результаты, ожидаемые для компонента С2, жестким кодированием их или читая их из файла.

Все это работает, если только между С2 и С3 нет никаких циклических зависимостей. В присутствии циклических зависимостей оба компонента должны быть полностью реализованы и индивидуально проверены до интеграции. Но даже и тогда циклические зависимости создадут кошмар тестирования. С большими циклами в структурном проекте тестирование интеграции должно быть выполнено как единая операция, называемая *тестированием «одним махом»*. Тестирование «одним махом» может быть успешно выполнено только для небольших программных решений. Это не очень разумно в современной разработке ПО.

Предположим теперь, что С2 и С3 были реализованы и индивидуально протестированы, но С1 должен быть еще разработан. Как мы должны интегрировать С2 и С3, чтобы объединенный экземпляр(конструкция) получал данные и другой контекст, которые стандартно давал бы компонент С1 и так, чтобы мы могли утверждать, что этот экземпляр работает таким образом, как ожидается при наличии С1? Чтобы сделать это, для С1 должен быть запрограммирован *тестовый драйвер***,** чтобы «управлять» интеграцией.

В целом интеграция требует написания дополнительного ПО, заглушек и драйверов, которые полезны только во время интеграции. Это дополнительное тестирующее ПО называется средствами тестирования или *«лесами для тестирования»*, по аналогии с временными лесами, используемыми в строительной индустрии.

Интеграция может проводиться *сверху вниз* (от корня иерархии зависимостей) или *снизу вверх* (от компонентов в листьях иерархии). Нисходящий подход требует реализации заглушек. Восходящий подход требует драйверов. В действительности интеграция редко следует только одному из этих подходов. Смешанный подход, иногда называемый *«из середины»*, является преобладающим.

Подобно интеграции, внедрение – не одноразовая операция. ПО внедряется своими версиями. Каждая *версия* объединяет ряд экземпляров (конструкций), которые предлагаются совместно и функционально полезны пользователям. Перед внедрением ПО *системно тестируется* разработчиками в реальных условиях. Оно иногда называется *альфа-тестированием.* За альфа-тестированием следуют *приемочные испытания* специалистами пользователя. Это иногда называется *бета-тестированием* (альфа- и бета-тестирование – термины, в большей мере используемые при тестировании системного ПО от имени продавцов ПО, в противоположность разработке прикладного ПО).

Кроме самой системы и приемочных испытаний, внедрение включает ряд других действий. Наиболее важное из этих действий – *обучение пользователей.* Практически обучение пользователей может начаться задолго до того, как система будет подготовлена к выпуску. Обучение совпадает по времени с производством *документации для пользователя.*

5.2.5. Процесс функционирования и сопровождения

Процесс функционирования представляет стадию жизненного цикла, когда программный продукт ежедневно используется, а работа предыдущей системы (ручной или автоматизированной) постепенно сокращается. *Постепенное сокращение* – обычно поэтапный процесс. В ситуациях, где это возможно и выполнимо, организация обычно использует новую и старую системы параллельно в течение некоторого времени. Это обеспечивает отход назад, если повое изделие не удовлетворяет бизнес-требованиям.

Процесс функционирования совпадает с началом сопровождения изделия. В программной инженерии сопровождение имеет несколько отличное значение по сравнению с обычным использованием этого слова. Во-первых, сопровождение – это не только незапланированная фиксация возникающих проблем. Сопровождение планируется и оплачивается на ранних стадиях жизненного цикла. Во-вторых, сопровождение включает развитие изделия. В некоторых итератив-

ных моделях жизненного цикла могут даже оказаться трудноразличимыми разработка и сопровождение.

Сопровождение обычно разделяется:

– на *корректирующее* (действия по обслуживанию) – выявление дефектов и ошибок, обнаруженных при работе;

– *адаптивное* – изменение ПО в ответ на изменения в вычислениях или в бизнес-среде;

– *совершенствующее* – развитие изделия путем добавления новых особенностей или улучшения его качества.

Стоимость сопровождения значительна за время жизни ПО. Она значительна потому, что изделие остается в действии в течение долгого времени. Большие системы предприятий настолько фундаментальны, что они сохраняются действующими с помощью использования любых доступных технологий «поддержки жизни». Такие системы называются унаследованными системами. Конечно, их следовало бы удалить, но для них нет никакой замены. Действительно, когда решение об *удалении* системы, в конечном счете, принято, это делается не потому, что унаследованная система больше не полезна для организации, а потому, что нет технических возможностей дальше сохранять ее в работе. Не должно быть сюрпризом, что сопровождение является главной причиной, почему это технически невозможно – через какое-то время сопровождение настолько уничтожает структурную ясность ПО, что оно дальше не может поддерживаться.

5.3. Модели жизненного цикла программного обеспечения

Жизненный цикл ПО определяет «что», но не «как» выполнять в процессе программной инженерии. Программная инженерия в значительной степени – социальное явление, определяемое внутренней организационной культурой предприятия. Предприятие может выбрать базовую **модель жизненного цикла,** но специфические особенности жизненного цикла и то, как работа будет сделана, уникальны для каждой организации и могут даже значительно отличаться от проекта к проекту. Это согласуется с наблюдениями, сделанными в ранее: программный продукт ***не изготавливается, он реализуется***. Процесс создания и эксплуатации ПО – не эксперимент, который может быть повторен много раз с той же самой степенью успеха.

Далее приведен краткий список причин, почему специфические особенности жизненного цикла должны быть приспособлены к организационной культуре и почему они отличаются от проекта к проекту.

Опыт программной инженерии, навыки и знания коллектива разработчиков (если их недостаточно, должно быть включено в процесс разработки и время в соответствии с «кривой обучения»).

Деловой опыт и знания (это намного более неприятно, чем предыдущий момент, потому что деловой опыт и знания легко не приобретаются).

Вид предметной области (необходимы различные процессы, чтобы совершенствовать бухгалтерский учет или систему контроля электростанции).

Изменения деловой атмосферы (изменения внешнеполитических, экономических, социальных, технологических и конкурентных факторов).

Внутренние деловые изменения (изменения в управлении, условиях работы, финансовое здоровье предприятия и т. д.).

Размер проекта (большой проект требует различных процессов, начиная с небольших; очень маленький проект может даже вообще не нуждаться в каких-либо процессах, поскольку разработчики могут сотрудничать и обмениваться информацией неофициально).

С учетом сказанного подходы к жизненному циклу ПО могут быть грубо разделены на две главные категории:

– «каскад (водопад) с обратной связью»;

– итеративный пошаговый.

5.3.1. Жизненный цикл «каскад с обратной связью»

Модель каскада (в зарубежной литературе часто используется термин «модель водопада») – традиционный жизненный цикл, реализованный и популяризируемый в 1970-х годах. Она успешно использовалась во многих больших проектах прошлого. Большинство этих проектов было связано с пакетными (то есть не диалоговыми) системами, реализованными на языке КОБОЛ. Сегодня жизненный цикл «водопад» используется менее часто.

Используя стадии жизненного цикла, принятые в этой главе, жизненный цикл «каскад с обратной связью» может быть представлен так, как изображено на рис. 5.2.

Это линейная последовательность стадий, из которых предыдущая стадия должна быть закончена прежде, чем может начаться следующая. Завершение каждой стадии отмечается подписанием документа для этой стадии проекта. Обратные связи (обратные стрелки на рис. 5.2) между стадиями возможны и даже целесообразны. Обратная связь показывает не документированное, но необходимое измене-

ние в более поздней стадии, которое должно быть завершено соответствующим изменением в предыдущей стадии. Такой возврат может (хотя это бывает редко) продолжаться до начальной стадии – анализа требований.

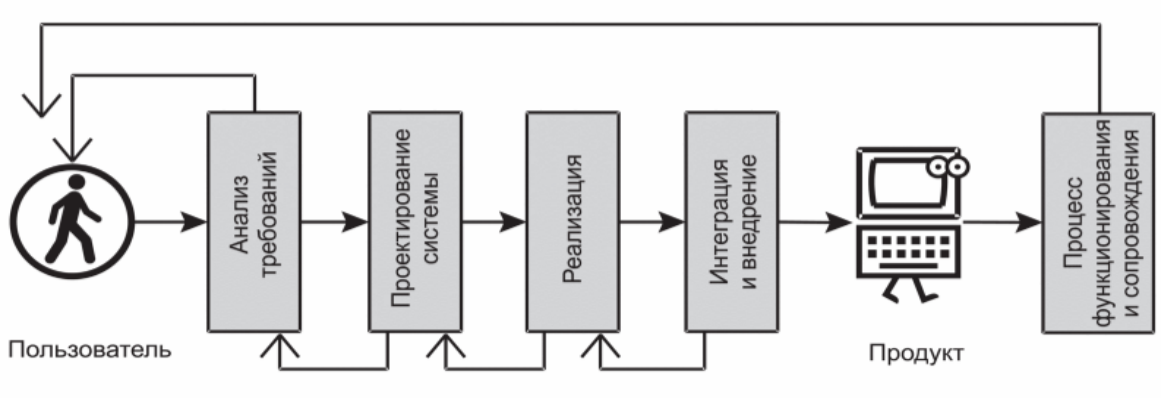


Рис. 5.2. Схема жизненного цикла ПО «Каскад с обратной связью»

Имеется много вариантов модели жизненного цикла «водопад». Один такой вариант допускает наложение стадий, то есть следующая стадия может начинаться прежде, чем предыдущая полностью закончится, будет задокументирована и подписана. Другой популярный вариант, иногда называемый моделью с опытными образцами, учитывает создание опытных образцов ПО на стадиях, предшествующих стадии реализации.

Критический момент для методов водопада – то, что они являются *монолитными,* они применяются при разработке по принципу «выполнять одновременно» ко всей системе в целом и задают единственную дату поставки системы. Пользователь подключается только на ранних стадиях анализа требований и заканчивает техническим заданием. Позже в течение жизненного цикла пользователь находится в полном неведении, пока изделие не может быть протестировано им до его внедрения. Поскольку *время задержки* между началом проекта и поставкой ПО может быть существенным (месяцы или даже годы), доверие между пользователями и разработчиками может быть утеряно, и разработчикам придется все сложнее защищать проект и оправдывать израсходованные ресурсы.

Использование опытных образцов в любом жизненном цикле является положительным, но оно дает особые преимущества для модели водопада, добавляя некоторую гибкость в ее монолитную структуру и смягчая риск поставки изделия, не отвечающего требованиям

пользователя. Опытный образец ПО – частный, «сляпанный наспех» пример решения проблемы. Из этого решения могут быть получены последовательные формы программного продукта.

В программной инженерии опытные образцы используются с большим успехом, выявляя и делая понятными требования пользователя к изделию. После того как опытный образец один раз будет использован в этом качестве, возникает вопрос, что делать с его ПО. Одна из возможностей состоит в том, чтобы забросить его подальше, как только цель подтверждения правильности требований будет достигнута. Однако опытный образец, созданный по требованиям анализа, может служить для улучшения проекта системы, а улучшенный опытный образец проекта может быть преобразован в конечный продукт.

5.3.2. Итеративный пошаговый жизненный цикл

Итерация в разработке ПО (в противоположность итерации в программе) – повторение некоторого процесса с целью улучшить программный продукт. Каждый жизненный цикл имеет некоторые элементы итеративного подхода. Например, обратные связи и наложения в модели водопада представляют своего рода итерацию между отдельными фазами, стадиями или действиями. Однако модель водопада не может считаться итеративной, потому что итерация означает движение от одной версии изделия к его следующей версии. Подход модели водопада монолитен, с формированием только одной заключительной версии изделия.

Итеративный жизненный цикл предполагает шаги – улучшенные или расширенные версии изделия в конце каждой итерации. По этой причине итеративная модель жизненного цикла иногда называется эволюционной или пошаговой.

Итеративный жизненный цикл также предполагает наличие конструкций – исполняемого кода, полученного в итерации. Конструкция – вертикальный срез системы. Это не подсистема. Область действия конструкции – вся система, но с некоторыми уменьшенными функциональными возможностями, с упрощенными пользовательскими интерфейсами, с ограниченной многопользовательской поддержкой, меньшим Конструкция – это нечто, что может демонстрироваться пользователю объемом работы и другими подобными ограничениями как версия системы, полученная в процессе продвижения к конечному продукту. Каждая конструкция фактически является новым шагом

по отношению к предыдущей. В этом смысле понятия конструкции и шага не различимы.

Итеративный жизненный цикл предполагает *короткие итерации* между шагами, в недели или дни, но никак не в месяцы. Это позволяет осуществлять непрерывное планирование и надежное управление. Работа, выполненная на предыдущих итерациях, может быть измерена и может дать ценную информацию для планирования выполнения проекта. Наличие работоспособных двоичных кодов, подлежащих сдаче в конце каждой итерации, позволяет надежно управлять проектом. При этом каждая итерация – «маленький водопад», состоящий из типичных стадий жизненного цикла. Различия лишь в масштабе, как сказано выше. Теперь пользователь может часто и полностью использовать основной цикл стадии функционирования и сопровождения. Уроки предыдущей итерации немедленно используются в следующей итерации. Пользователь, вооруженный опытом использования предыдущей конструкции, может быть очень полезен в уточнении требований для следующей итерации. Текущий проект конструкции – отправная точка для проекта системы в следующей итерации. Внедрение итерации 2 продукта является началом итерации 3 и т. д.

Классическая модель итеративного жизненного цикла – спиральная модель. Современным представителем итеративного жизненного цикла является

IBM Rational Unified Process (RUP), который создан из Rational Unified Process (RUP)]. Более современные представители итеративной модели жизненного цикла – Model Driven Architecture (MDA) и agile development – быстрая разработка.

5.3.3. Спиральная модель

Спиральная модель в действительности является *метамоделью*, в которой могут содержаться все другие модели жизненного цикла. Модель состоит из четырех секторов диаграммы в декартовых координатах (рис. 5.3). Сектора, следующие: планирование, анализ рисков, инженерия и оценка проекта клиентом. Первая петля спирали начинается в секторе *планирования* и связана с начальным сбором требований и планированием проекта. Затем проект входит в сектор *анализа рисков*, где проводится анализ стоимости/ выгоды и угроз/ благоприятных случаев, чтобы принять решение «да–нет» относительно того, следует ли входить в сектор *инженерии* (или отказаться от проекта как слишком опасного). Сектор инженерии – это то, где происходит

разработка ПО. Результат этой разработки (конструкция, опытный образец или даже конечный продукт) подвергается *оценке клиентом*, после чего начинается вторая петля спирали.

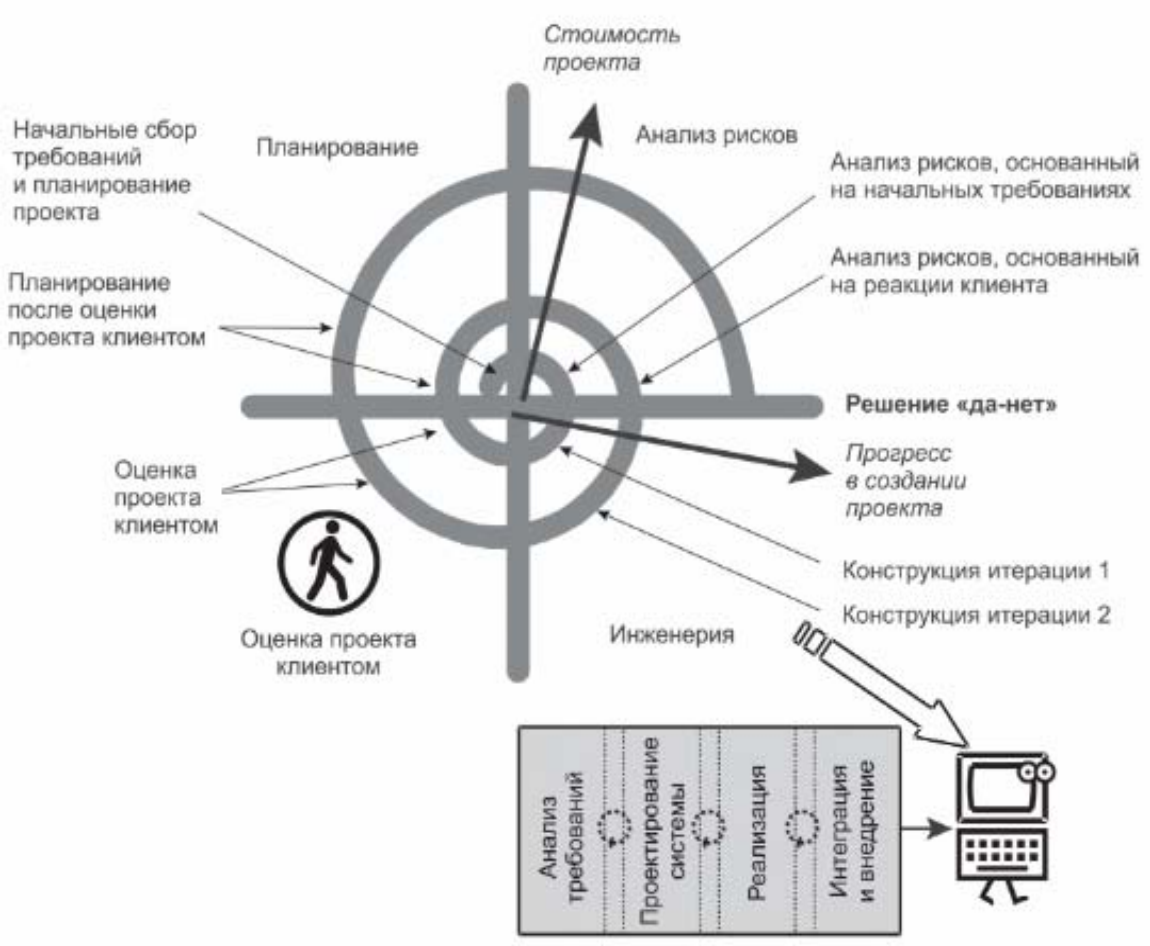


Рис. 5.3. Спиральная модель жизненного цикла ПО

Фактически спиральная модель имеет отношение к разработке ПО только в одном из этих четырех секторов инженерии.

Акцент на повтор в планировании проекта и оценке проекта клиентом дает всему этому явно итеративный характер. Анализ рисков уникален в спиральной модели. Частые исследования рисков позволяют провести раннюю идентификацию любых появляющихся рисков в проекте. Риски могут быть внутренние (находящиеся под управлением организации) и внешние (риски, которыми организация не может управлять). В любом случае задача анализа рисков состоит в том, чтобы смотреть в будущее, и если ясно, что риски неустранимы, проектирование должно быть прекращено безотносительно к затратам, уже израсходованным.

Каждый фрагмент петли в пределах сектора инженерии может быть итерацией, заканчивающейся созданием конструкции**.** Любой последовательный фрагмент петли в направлении наружу представляет собой шаг.

Возможны и другие интерпретации фрагментов петли инженерии. Например, вся петля спирали может быть связана с анализом требований. В таком случае фрагмент петли инженерии может быть связан с моделированием требований и созданием раннего опытного образца, чтобы выявить требования. С другой стороны, спиральная модель может содержать монолитную модель водопада. В таком случае будет только одна петля спирали.

Спиральная модель – скорее модель ссылок или метамодель для других моделей. Привлекательная и реалистическая, насколько это возможно, она не может быть перенесена на конкретный задокументированный жизненный цикл, который организации используют при разработке ПО. Спиральная модель – «способ мышления» относительно принципов разработки ПО.

5.3.4. Быстрая разработка ПО с короткими итерациями

Процесс **быстрой разработки ПО,** предложенный в 2001 г. некоммерческой организаций энтузиастов Agile Alliance, является смелым новым подходом к производству ПО. В Manifesto of Agile Alliance дух быстрой разработки представлен четырьмя рекомендациями:

1. Индивидуальные характеристики и взаимодействия процессов и инструментальных средств.
2. Рабочее ПО с полной документацией.
3. Сотрудничество с клиентами после заключения контракта.
4. Определение, что нужно изменить после планирования.

Быстрая разработка подчеркивает, что производство ПО – творческая деятельность, которая зависит от сотрудничества людей и коллективов гораздо больше, чем от различных процессов, использования инструментальных средств, документации, планирования и других формальных операций. Быстрая разработка подтверждает принцип, что «большое ПО делают большие коллективы людей», все остальное вторично. «Люди» включают всех участников создания проекта – разработчиков и клиентов. В отличие от других процессов создания ПО, при быстрой разработке клиенты (пользователи и собственники системы) тесно работают с коллективом разработчиков на протяжении всего жизненного цикла, а не только в его начале. Постоянная обратная связь клиента позволяет легче подписать формальный кон-

тракт на все изделие. Интенсивное сотрудничество облегчает также получение той части документации, которая обеспечивает передачу знаний.

Несмотря на все эти «революционные» суждения быстрая разработка хорошо выглядит среди других итеративных жизненных циклов. Быстрый жизненный цикл не может использовать терминологию обычных стадий жизненного цикла, однако на самом деле его терминология соответствует обычному циклу анализа, проектирования, реализации и внедрения.

«Обычный» анализ требований заменен в быстрой разработке *историями пользователей*, где клиент отмечает особенности, которые, по его мнению, система должна поддерживать. Коллектив разработчиков оценивает, сколько потребуется времени для реализации каждой истории, и сколько реализация каждой истории будет стоить. После этого клиент может выбрать те истории, которые будут реализованы на первой и последующих итерациях. «Обычные» виды проектирования систем и их реализация заменены в быстрой разработке комбинацией приемочных испытаний, рефакторинга и управляемой тестированием разработки. *Приемочные испытания* – это специальные программы, через которые разрабатываемая прикладная программа должна пройти, чтобы быть принятой клиентами. Этот процесс называется **«разработкой, управляемой тестированием»** (Test-Driven Development – TDD) или *программированием намерений* (Intentional Programming – IP) – разработчик программирует свои намерения по приемочным испытаниям до того, как использует их. Этому подходу сопутствует частое использование **рефакторинга** (refactorings) – усовершенствования в структуре системы без изменения ее поведения.

Быстрая разработка поддерживает и другие концепции типа **парного программирования** и *коллективной собственности.* Все программирование выполняется парами программистов – двумя программистами, работающими вместе на одной рабочей станции. Один программист пишет код, а другой наблюдает за процессом и задает вопросы. Роли меняются всякий раз, когда один из программистов хочет «поставить точку с помощью клавиатуры». Пары программистов также меняются местами, по крайней мере, один раз в день. Результатом является *коллективная собственность.* Никто индивидуально не владеет кодом. Считается, что высокий дух коллективизма и желание выдать продукт перевесят любые требования индивидуальной ответственности.

«Обычные» интеграция и внедрение заменены в быстрой разработке *непрерывной интеграцией* и *короткими циклами.* Пары программистов могут экспортировать свой код и по своему желанию объединять его с остальной частью. Более того, одну и ту же часть кода может импортировать и работать над ней более одной пары программистов. Это может привести к конфликту, когда коллектив, который хочет экспортировать и сдать свой код, обнаруживает, что другой коллектив сделал ранее несовместимый код. Такие конфликты между участвующими в разработке коллективами должны быть урегулированы.

Быстрая разработка не означает плохое планирование. Фактически даты внедрения тщательно планируются. Каждая итерация обычно планируется так, чтобы закончить работу за короткий цикл продолжительностью в две недели. Продукт в конце двухнедельного цикла – пробный вариант для оценки клиентом. Основная поставка продукта в производство является результатом приблизительно шести двухнедельных циклов.

Быстрая разработка отличается больше в методах, чем в подходе к итеративной разработке. Ее главный представитель – **extreme Programming** (XP). Как и с MDA-подходом, будущее покажет, сможет ли быстрая разработка применяться к большим и сложным системам. Главная опасность, стоящая перед сторонниками быстрого жизненного цикла – риск окончания работы с неудачно *созданной и принятой моделью*, в которой ПО слеплено без учета заданных требований к проекту.

5.4. Инструментальные средства моделирования систем

Как было отмечено выше, программная инженерия сродни моделированию. Следовательно, диапазон *инструментальных средств моделирования систем* охватывает все средства, которые поддерживают инженеров на всех этапах жизненного цикла ПО.

В настоящем пособии нет необходимости подробно рассматривать возможности средств моделирования, поскольку предполагается, что этот материал студенты-магистранты изучили ранее в рамках бакалаврской подготовки. Поэтому мы ограничимся обзором этих средств.

Инструментальные средства моделирования систем основаны на использовании различных языков моделирования. Язык является средством связи между людьми. ПО создано людьми и для людей.

Поэтому нужно, чтобы модели ПО были коммуникабельны. Моделирование ПО требует наличия языка как средства выражения процессов разработки ПО и созданных продуктов. Язык должен быть универсальным (должен существовать его. Вообще-то говоря, ни UML, ни любой другой язык моделирования не достиг еще статуса языка для полной разработки жизненного цикла. В частности, запутанность программирования мешала искушению реализовать суждение «один язык годен на все случаи жизни». Программисты работают с различными языками, которые лучше удовлетворяют их по той или иной причине. Это означает, что при переходе к программированию UML-модели должны быть переведены на язык программирования. Кроме того, когда выполняется детальное проектирование для различных предметных областей и различных технологий, UML имеет недостаточную мощь и разнообразие, чтобы обслужить все цели проектирования. Специализированные UML-профили**,** типа UML для моделирования сети или UML для бизнес-моделирования, несколько смягчают эту проблему, но есть еще много нерешенных вопросов.

«Одно изображение стоит тысячи слов». Языки моделирования ПО используют эту старую истину. Эти языки визуальны, но с добавлением текста. Иногда графическая модель должна быть далее разъяснена, чтобы избежать неверного толкования. В других случаях одна графическая модель не дает целую картину, и в тексте обеспечивается дополнительная информация. В некоторых случаях текст доминирует над визуальным представлением. Интересно, что определение требований в начале жизненного цикла и программирование в конце его преобладающе состоят из текста. Средние стадии анализа и проектирования главным образом визуальны.

Следовательно, спецификации ПО состоят из большого числа визуальных моделей, снабженных текстом, определяющих различные стадии разработки продукта и обеспечивающих различные (часто перекрывающиеся) точки зрения на продукты ПО. Грубая классификация моделей различает модели анализа, проектирования и реализации. Каждая модель состоит из одной или большего количества диаграмм и всевозможной дополнительной информации, размещенной в хранилище проекта.

Диаграмма – графическое представление модели, которое отображает символы ПО. Каждая диаграмма содержит различные аспекты модели или представляет модель на различных (более или менее детальных) уровнях абстракции. Основная классификация различает

диаграммы, содержащие состояние системы, ее поведение или изменения состояния.

Диаграммы и текстовые описания моделей размещаются в хранилище проекта. Хранилище – главный компонент любого инструмента визуального моделирования. Это – БД продуктов проекта. Термин «база данных» (БД) подразумевает сохраняемое и интегрированное размещение данных, допускающее параллельный доступ к ним многих пользователей. Разработка ПО – совместное усилие многих разработчиков, и все они должны иметь параллельный доступ к моделям разработки. Большинство инструментальных средств визуального моделирования использует всю мощь технологии БД или даже применяет коммерческие системы управления БД (СУБД), чтобы реализовать их хранилища. Традиционно средства визуального моделирования называются CASE-инструментами (computer assisted software engineering – автоматизированная программная инженерия).

5.4.1. Язык структурного моделирования

Структурное моделирование было популярно в конце 1970-х годов в результате широкого использования структурного программирования. Структурное программирование представляет собой программирование без использования операторов *goto* с циклами и операторами *if* как основными управляющими, конструкциями и с нисходящим подходом к проектированию программ.

Структурное моделирование – нисходящий функционально-ориентированный подход к разработке ПО, который расчленяет систему на ряд взаимодействующих в общем процессе функций и известный как функциональная декомпозиция. Функциональная декомпозиция – пошаговый процесс, который основан на хорошем использовании абстракции. Модели системы представляются на ряде уровней абстракции, начинающихся с вершины – контекстной диаграммы, дальнейшей детализации и заканчивающихся на простейших функциональных модулях, определенных в деталях и существенных для реализации.

Структурное моделирование выражает монолитный и процедурный характер систем прошлого, например, стиля КОБОЛ. Нисходящий подход структурного программирования использовался на стадиях анализа и проектирования, в результате чего были разработаны различные методы структурного анализа и проектирования. Структурное моделирование задает диапазон технологий визуализа-

ции, наиболее популярными из которых были DFD (data flow diagrams – диаграммы потока данных), ERD (entity-relationship diagrams – диаграммы «сущность-отношение») и диаграммы структур. Структурное моделирование обычно нацелено на функциональную декомпозицию, представленную в DFD. ERD и диаграммы структур используются как поддерживающие технологии. DFD имеют три основных элемента моделирования: процессы, потоки данных и информационные хранилища.

Технология моделирования данных ERD широко используется за пределами структурного моделирования для представления структур баз данных. ERD строятся, с использованием двух графических элементов: сущности и отношения. И сущности, и отношения могут содержать атрибуты.

5.4.2. Язык объектно-ориентированного моделирования

Моделирование современных (объектно-ориентированных) систем производится с помощью Unified Modeling Language (UML) – унифицированного языка моделирования. UML «является языком для определения, визуализации, создания и документирования компонентов систем ПО, а также для бизнес-моделирования и других систем, не являющихся ПО. UML представляет собрание лучших технических методов, которые доказали свою эффективность в моделировании больших и сложных систем. Это метод детального описания архитектуры системы, облегчающий процессы создания и сопровождения системы. В настоящее время UML 2.4.1 принят в качестве международного стандарта ISO/IEC 19505-1, 19505-2.

Объект, согласно объектно-ориентированному подходу, определен как часть ПО, которая имеет состояние, поведение и индивидуальность.

Состояние объекта определяется значениями его атрибутов.

Поведение объекта определено сервисами (операциями), которые объект может выполнять, когда он вызывается другими объектами.

Индивидуальность объекта – свойство объекта, когда любые два объекта в системе рассматриваются различными, даже если они обладают теми же самыми значениями атрибутов и операциями. Это означает, что два объекта могут быть равны, но они никогда не идентичны. По этой же причине объект может существенно изменять свое состояние и поведение, но при этом оставаться тем же самым объектом с той же самой индивидуальностью.

В соответствии с вышеупомянутыми характеристиками объектов существует классификация объектно-ориентированных моделей, включающая ***модели состояний*, *модели поведения* и *модели изменения состояний.*** Каждая из этих моделей представляется одной или несколькими диаграммами.

Объектно-ориентированное UML-моделирование имеет дело с диаграммами классов, но в первую очередь определяется диаграммами сценариев использования. Модель сценариев использования – ориентир для всех других моделей. Другие модели обращаются к модели сценариев использования, чтобы выяснить требования пользователя и/ или проверить, соответствуют ли они требованиям пользователя. Диаграммы классов представляют и состояние, и поведение системы. В конечном счете модели классов определяют главный подход к программированию.

Диаграммы классов визуализируют классы объектов, содержание их атрибутов и операций, а также отношения между классами. Имеются три вида

Диаграммы сценариев использования обеспечивают простую визуализацию сценариев использования, их отношений и акторов (действующих субъектов), взаимодействующих со сценариями использования. Однако реальная мощь сценариев использования не в визуальных моделях, а в текстовых определениях требований пользователя, которые должны быть обеспечены этими диаграммами.

Диаграммы взаимодействия – основная технология моделирования поведения (уровня проектирования) в UML. Они представляют передачу сообщений в системе. Имеются два вида диаграмм взаимодействия: диаграммы последовательности действий и диаграммы сотрудничества (связи).

Диаграммы состояний охватывают состояния объектов и действия, которые приводят к переходам между состояниями объектов. Они предназначены для отдельных классов, хотя могут также использоваться и для моделирования изменений состояния в сложных элементах модели, таких как пакеты или даже целая система.

Диаграмма деятельности – конечный автомат, который представляет выполняемые в системе вычисления. Как правило, диаграмма деятельности дополняет реализацию операций или сценарий использования.

Диаграммы выполнения – модели для физической реализации системы. Они показывают компоненты системы, их структуру и зави-

симости, а также их размещение в узлах компьютерной системы. Имеются два вида диаграмм выполнения: диаграммы компонентов и диаграммы размещения.

В UML визуальное моделирование обеспечивается так называемыми классификаторами. Классификатор – это элемент модели, который описывает поведение или структуру системы, и обычно имеет визуальное представление. Примеры классификаторов включают class (класс), actor (актор), use case (сценарий использования), relationship (отношение).

Спецификация UML определяет шесть видов диаграмм: state structure (структуры состояний), use case (сценариев использования), interaction (взаимодействия) sequence diagram) (последовательность деятельности), а также диаграммы implementation (выполнение).

5.5. Инструментальные средства программной инженерии

5.5.1. Роль инструментальных средств программной инженерии

Базовый термин для инструментальных средств программной инженерии – **автоматизированная программная инженерия** (computer assisted software engineering – CASE). Однако практически смысл CASE-средств обычно ограничивается возможностями визуального моделирования для анализа и проектирования систем. Хотя большинство таких инструментальных средств имеет возможность прямого проектирования (формирование кода) и обратного проектирования (от существующего кода к визуальным моделям), они обычно не составляют законченную среду программирования. Инструментальное средство, которое всесторонне обеспечивает программиста, называется **интегрированной средой разработки** (integrated development environment – IDE).

CASE и IDE не охватывают все инструментальные средства, используемые в процессах создания ПО. Жизненный цикл ПО требует таких инструментальных средств *управления проектом*, которые планировали бы и управляли ресурсами проекта. Он также требует средств *управления изменениями,* чтобы те учитывали недостатки, обеспечивали корректировку и заботились о полном сопровождении ПО. Будучи по своей сути деятельностью коллектива людей, программная инженерия нуждается в технологии *управления конфи-*

гурацией ПО. В соответствии с этим ниже представлены следующие группы инструментальных средств программной инженерии:

– инструментальные средства управления проектом;

– инструментальные средства моделирования систем (CASE);

– среды формирования ПО (IDE);

– инструментальные средства управления изменениями и конфигурацией.

Инструментальные средства программной инженерии – крупный и достаточно конкурентоспособный бизнес. В то время, когда читатели получат эту книгу, используемые в промышленности инструментальные средства, упомянутые в этой главе, могут быть, а могут и не быть в представленной здесь форме. Читатель должен попытаться выполнить с помощью Интернета соответствующие исследования самой последней информации.

Инструментальные средства управления проектом

Управление проектом объединяет множество инструментальных средств, методов и технологий, связанных с планированием проекта и управлением интегрированным процессом.

Управление проектом – нечто вроде распределения и управления бюджетом, временем и людьми. Главные вопросы здесь:

Сколько будет стоить разработка системы?

Сколько потребуется времени на разработку системы?

Какие люди нам нужны для разработки системы?

Старый принцип гласит, что, если вы не сможете спланировать что-то, вы не сможете это и сделать. Но планирование не может быть «высосано из пальца». Оно требует некоторых начальных знаний того, что требуется для разработки системы в этой организации, с этими людьми, с этими ресурсами, в этой организационной культуре и т. д. Чтобы иметь это, нужно знать прошлое организации. Организация должна оценить, что происходило с предыдущими проектами. Она должна определить из предыдущих проектов *метрики* (систему показателей). Это приводит к следующему принципу: «если вы не знаете ваше прошлое, вы не можете должным образом планировать и ваше будущее».

Управление проектом требует использования инструментальных средств для эффективного планирования и регулирования показателей проекта, для оценки проектных затрат, для сбора метрик и т. д. Современные инструментальные средства часто включаются в интегрированные средства поддержки управления. Эти средства объеди-

няют обычное управление проектом со стратегическим планированием, бизнес-моделированием, портфельным управлением, управлением документацией, технологическим процессом и т. д. Такие интегрированные среды представляют единое универсальное инструментальное средство для управления проектами в более широком контексте выявленных стратегических инициатив, обеспечения тактического управления, выполнения ежедневных задач и управления людьми. Интегрированные среды управления могут касаться не только предприятия, но и поставщиков, клиентов и других деловых партнеров.

Лучшие представители управления большими проектами используют Web-технологии, которые позволяют осуществлять динамическое, интерактивное и синхронное управление большими коллективами и процессами. Многие традиционные инструментальные средства управления проектами имеют Web-версии.

5.5.2. Планирование и управление проектом

Инструментальные средства для **планирования и управления** показателями проекта помогают в подготовке графиков операций. **Сетевые графики** операций задают события, которые должны произойти до того, как начинается операция, определяют продолжительность каждой операции и ее сроки, распределяют людские и другие ресурсы на все операции и т. д. Два традиционных вида графиков операций – **метод критического пути** (critical path method – CPM) и **диаграммы Ганта** (Gantt charts). Известное расширение CPM – это **система планирования и руководства разработками** (Program Evaluation and Review Technique – PERT).

Инструментальные средства управления проектами могут формировать график операций, как только будет обеспечена необходимая проектная информация. Требуемая проектная информация включает определение операций (задач), их продолжительность и ресурсы, предшествующие события и события, которые являются следствиями операций, и т. д. Проектную информацию можно разместить в тексте или ввести непосредственно в график.

Инструментальные средства управления проектами могут также автоматически преобразовывать графики (например, PERT в диаграммы Ганта и наоборот) и представлять графики, подчеркивающие различные точки зрения и с различными деталями. Например, отдельные графики могут быть сделаны для всего проекта и для отдельных сотрудников, работающих над проектом. Графики могут отобра-

жаться, используя различные временные шкалы, показывая (или не показывая) определенные человеческие и другие ресурсы, продолжительности операций и/ или даты их начала и окончания.

Инструментальные средства управления проектами регулируют и перестраивают планы всякий раз, когда происходят изменения в операциях или ресурсах. Планы корректируются, одновременно заново пересчитываются даты начала и конца операций с информацией о последних изменениях. Они могут быть повторно рассчитаны, чтобы учесть влияние добавления ресурсов в проект. Это позволяет менеджерам понять, может ли дополнительный ресурс дать лучшие результаты (иногда организаторские накладные расходы, связанные с добавлением нового ресурса, могут увеличить время разработки).

5.5.3. Управление проектированием и реализацией с учетом основных показателей

Организации функционируют в конкурентной и динамичной среде. Чтобы выживать и успешно конкурировать, организации должны постоянно приспосабливать свои показатели, цели, планы, а также проекты разработок к изменениям в потребностях клиентов, к новым технологиям, изменениям внешних факторов и т. д. В этих условиях управление проектом должно отвечать требованиям совершенствования результативности, эффективности и качества.

Неудивительно, что ряд инструментальных средств управления проектами включен в более общие программные средства, чтобы обеспечить широкое разнообразие управленческих операций. В частности, они обеспечивают управление проектом в контексте совершенствования **управления реализацией** и стратегического планирования. Они помогают в управлении людьми и коллективами в вопросах планирования, отслеживания результатов и достижения бизнес-целей. Акцент перемещен от проектов к людям. С людьми нельзя обращаться как с простыми «ресурсами», распределенными по операциям или задачам. Людьми управляют, задавая им бизнес-цели и прослеживая результаты.

Унификация управления проектом с организацией совместной работы и информационного обеспечения на основе Web-технологии. Многие проекты ПО являются распределенными – они охватывают большое количество людей и выходят за обычные организационные границы. Поэтому связь между участниками создания таких проектов должна использовать Web-технологии. Управление проектом становится «компонентом» сотрудничества *специалистов*

в сфере информационных технологий на основе Интернета. Специалисты принадлежат различным распределенным коллективам. Чтобы обеспечить выполнение проекта, они должны планировать, общаться, сотрудничать, принимать решения, распределять обязанности и т. д. по Интернету.

Управление совместной работой включает поддержку передачи спорных вопросов заинтересованным сторонам проекта, определения и задания соответствующих мероприятий, формирования документации. Чтобы напоминать о событиях и выполняемых операциях, могут быть установлены правила совместной работы. Наряду с обеспечением сотрудничества участников проекта управление проектом на основе Web-технологии обеспечивает легкий доступ и передачу документации и других информационных материалов типа Web-страниц и сообщений по электронной почте. Инструментальные средства такой категории используют общедоступные библиотеки документации и другие источники информации, то есть, выполняют то, что сегодня известно как информационное обеспечение. Чтобы информировать участников создания проекта об его изменениях, широко используется механизм подписки на рассылку информации.

Среда eRoom, разработанная фирмой Documentum, является примером инструментального средства на основе Web-технологии, которое объединяет управление проектом с организацией совместной работы и информационным обеспечением.

Унификация управления проектом на основе портфельной Web-технологии. Объединение управления проектом с организацией сотрудничества и информационным обеспечением, все это с использованием Интернета, является важным шагом в планировании и составлении графиков. Этот шаг является *многопроектным* и *с ограниченными ресурсами.* Инструментальные средства, поддерживающие такое планирование и составление графиков, концентрируются на совместных ресурсах и их распределении, а также на оценке общего времени. Этот процесс предоставления предприятию возможности распределить ресурсы и инвестиции в соответствии с основными показателями называется управлением портфелем проектов предприятия или просто портфельным управлением.

Инструментальные средства, которые объединяют управление проектом с портфельным управлением предприятия, обычно являются комбинацией целого ряда связанных средств типа управления коллективами и ресурсами, связями и совместной работой, документооборотом и информационным обеспечением. Инструментальные средства

этой категории типа eProject Enterprise фирмы eProject или Primavera Enterprise Product Suite фирмы Primavera Enterprise обеспечивают среду разработки на основе Web-технологий:

– для обсуждения проектов, назначения работ и их отслеживания;

– составления графиков работы коллективов и таблиц учета рабочего времени;

– проведения электронных конференций для достижения согласия и выработки консультативных решений;

– распределения проектных документов и поддержания их версий;

– автоматизации технологического процесса и отслеживания решений;

– обычного планирования и составления графиков с использованием диаграмм Ганта, CPM и PERT;

– формирования шаблонов документов и образцов для выбора лучших методов;

– установления соответствия проектов стратегическим планам и бизнес-моделям;

– управления индивидуальными и распределенными ресурсами;

– распределения возможностей человеческих и других ресурсов по проектам;

– формирования взаимных проектных резюме и индикаторов состояния.

5.5.4. Интеграция управления проектом с метриками

Метрики в программной инженерии не имеют никакого отношения к Европейской метрической системе. Метрики – это порядок количественной оценки процессов разработки ПО и созданных продуктов с тем, чтобы собранная информация могла помочь в будущем планировании проектов.

Несмотря на их важность, в большинстве проектов метриками пренебрегают. Ими жертвуют под ежедневным давлением необходимости выполнить текущий проект. Метрики требуют ресурсов и их перемещения от непосредственной разработки к сбору системы показателей – жестокое требование! Эта работа может также привести к требованию остановки в некоторых областях, хотя сам проект будет и на правильном пути.

Поскольку сбор метрик – дорогое удовольствие, иногда отвергаемое разработчиками и менеджерами, использование удобных и информативных инструментальных средств получения системы показателей приобретает особую важность. Инструментальные средства

получения метрик должны обеспечить легкий доступ к информации. Всякий раз, когда это возможно, информация должна автоматически получаться из моделей разработки и программного кода.

Анализ кода структурного проекта – важный пример того, как управление проектом может быть дополнено критически важными метриками. Структурное проектирование отвечает за *возможность сопровождения* (supportability) системы, то есть ее понятность, удобство сопровождения и масштабируемость. Поэтому важно оценить возможность сопровождения системы как на ранних стадиях жизненного цикла ПО, так и на поздних, когда должно быть оценено влияние изменений существующего кода. Метрики нацелены на определение мест минимизации зависимостей компонент с тем, чтобы улучшить общую стабильность и качество разрабатываемой системы.

5.5.5. Интеграция управления проектом с управлением рисками

От управления проектом и метрик есть только небольшой шаг к анализу рисков проектов. Риски определяются как «потенциально неблагоприятные обстоятельства, которые могут повредить процессу разработки и качеству продуктов». Риски – каждодневные события, которые сопровождают любую человеческую деятельность. Поэтому неудивительно, что управление рисками – отдельная отрасль теории управления.

Управление рисками – операция принятия решения, которая оценивает воздействие рисков (неопределенностей) на решения. Она оценивает распределения возможных проектных результатов по отношению к вероятностям достижения этих результатов. Приняв приемлемый уровень допущения риска, она оценивает вероятность, с которой произойдет желаемый результат.

Управление рисками проектов использует то же самое мышление, которое применяется в исследованиях рисков финансовых портфелей. Простой метод финансового планирования состоит в том, чтобы взять начальный портфель и оценить будущий доход, применяя средние нормы процентных ставок на возвращение активов. Использование средних возвращаемых величин может дать в лучшем случае средние результаты. Лучшие результаты могут быть получены путем использования теории вероятностей, принимая во внимание различные шансы. Шансы определяются различными функциями распределения вероятности, а диапазон возможных результатов вычисляется

с помощью моделирования. Управление рисками проекта использует подобные подходы.

Полагая, что желаемые проектные результаты определены, типичные шаги анализа риска, следующие:

1. Идентифицировать риски и представить их, используя диапазоны возможных величин. Это делается заменой значений риска выбранной функцией распределения вероятности типа распределений Пуассона, равномерного, хи-квадрат, треугольного и т. д.

2. Использовать моделирование типа метода Монте-Карло или Латинского гиперкуба, чтобы получить возможные результаты и вероятности этих результатов.

3. Принять решение на основе графических результатов (гистограммы, интегральные кривые и т. д.), использования персональных суждений и приемлемого уровня допущения риска.

Инструментальные средства, которые объединяют управление проектом с управлением рисками, часто реализуются как дополнения к обычным инструментальным средствам управления проектами (типа Microsoft Project), дополнения к электронным таблицам (типа Microsoft Excel) или как приложения БД (то есть, например, для Microsoft Access). Например, ряд программных продуктов, называемых @Risk от Palisade, состоит из дополнений к Microsoft Project и Excel. С другой стороны, Risk Radar[1 м] – продукты от ICE (Integrated Computer Engineering – Интегрированная компьютерная разработка) являются приложениями к Access.

Современные инструментальные средства моделирования систем – средства рабочей группы, которые обеспечивают многих разработчиков доступом к распределенному хранилищу продуктов разработки, импортом продуктов в локальные рабочие станции, работой на этих продуктах и возвращением их назад в хранилище.

Инструментальные средства моделирования систем сосредоточены на анализе требований и стадиях проектирования системы. Они обычно включают возможность как формирования исходного кода (прямое проектирование) на основании моделей, так и обратное проектирование моделей, исходя из кода. Одной из основных особенностей инструментальных средств моделирования является урегулирование зависимостей между требованиями пользователей (выраженных в виде текстовых предложений) и представление концепций визуального моделирования, отображающих эти требования.

Визуальное моделирование использует унифицированный язык моделирования (Unified Modeling Language – UML). Он вполне доста-

точен для всех задач моделирования, связанных с бизнес-требованиями, состоянием системы, ее поведением и динамическими изменениями состояния. Другие проблемы моделирования, типа моделирования БД или проекта, использующего Web-технологии, вероятно, потребуют специализированного языка моделирования или специально разработанного варианта UML (UML-профиля).

5.5.6. Управление требованиями

Требования – это текстовые предложения в пределах технического задания. Обычно имеется одно техническое задание на один сценарий использования – use case. Требования определяются на различных уровнях абстракции, и между ними имеется иерархическая согласованность. Описательное имя сценария использования также является требованием.

Инструментальное средство моделирования системы должно облегчить сотрудничество коллектива, обеспечить способы написания технических заданий. Идентификацию различных категорий требований в документах, управление их изменениями и формирование требований, доступных всем разработчикам. Универсальный доступ к требованиям разработчиков подчеркивает важность этих требований в жизненном цикле разработки. Они используются аналитиками, чтобы выполнить анализ модели, проектировщиками, чтобы определить структуру системы и детальную спецификацию проекта, испытателями, чтобы разработать контрольные примеры, и менеджерами проекта, чтобы разработать планы и бюджет.

Требования, определенные в техническом задании, размещаются в *хранилище* инструментального средства моделирования. Они могут быть отражены на экране и модифицированы разработчиками. Изменения названий требований автоматически отражаются в техническом задании и наоборот. Инструментальное средство должно поддержать навигацию между связанными требованиями и между требованиями и другими продуктами моделирования.

Типичный способ показа требований для манипуляций – формат электронной таблицы. Этот знакомый всем формат позволяет легко задавать значения различных свойств (атрибутов). Свойства требования определяют такие характеристики, как статус требования, трудность его реализации, стабильность (то есть, может ли оно изменяться), кто владеет этим требованием, кем оно назначено, когда оно было создано и/ или обновлено.

Интеграция управления требованиями с визуальным моделированием UML должна поддерживать двунаправленные ассоциации между требованиями (на различных уровнях) и визуальными элементами (продуктами) модели.

использования, классы, компоненты, методы и т. д.) усиливают ценность проекта. Этот тип инструментального средства – *«все в одном»* – старается включить управление требованиями во все основные процессы жизненного цикла ПО от моделирования требований до формирования ПО (прямое проектирование).

Интеграция требований и других элементов моделирования требует умелого представления моделей на различных уровнях детализации и четкой навигации между элементами. Имея возможность создания в проекте большого количества диаграмм и элементов модели, легко потеряться в «пространстве проекта». Те, кто обращал внимание на изменения в программе «Проводник», мог бы заметить особенность *представления пиктограмм*, которая позволяет пользователю помещать картинки на папки как напоминание об их содержании. Представление пиктограмм также формируется автоматически для папок, которые содержат изображения, показывая уменьшенные рисунки на изображениях папок. Пользователи не должны просматривать папку, чтобы увидеть ее содержание, так как вид пиктограммы уже дает хорошее представление об этом. Подобный подход может быть включен в инструментальные средства моделирования, чтобы избежать ненужного просмотра диаграмм и просто напомнить проектировщику относительно содержания диаграммы.

Кроме создания пиктограмм или использования подобных технологий существенно может улучшить управляемость системы группировка связанных требований и элементов модели в UML-пакеты (возможно, в пакеты различных уровней).

5.5.7. Визуальное UML-моделирование

Инструментальные средства для визуального **UML-моделирования** имеют диапазон от простых (только графических) средств до средств на основе хранилищ, поддерживающих коллективную работу и позволяющих настраивать графические представления. Наиболее сложные средства имеют хранилище на основе коммерческой БД, обеспечивают полную интеграцию моделей, интерфейс к другим инструментальным средствам программной инженерии, поддерживают различные версии продуктов и моделей проекта, формируют код для различных языков программирования и БД и т. д.

Все эти проблемы подробно рассмотрены в соответствующих главах этой книги.

Инструментальные средства **визуального UML-моделирования** весьма похожи на подход для реализации GUI (Graphical User Interface – графический пользовательский интерфейс). Они состоят из области графического моделирования и соответствующей панели инструментов, содержащей элементы моделирования. Окно навигационного браузера (типа проводника) помогает в навигации между различными диаграммами и концепциями моделирования в пределах проекта. Есть также окна документации, позволяющие вводить текстовые описания для элементов моделирования. Могут быть запущены чувствительные к контексту окна спецификаций, чтобы войти в детальные спецификации (включая текстовые описания) для элементов моделирования. Все диаграммы и спецификации размещены в единственном хранилище для того, чтобы разработчики могли трудиться в многопользовательском режиме.

Одним из средств визуального моделирования инструментального средства является Poseidon фирмы Gentleware. Poseidon основан на ArgoUML. Интересная особенность Poseidon заключается в том, что оно обслуживает документацию проекта с помощью модели, находящейся в блокноте с закладками (в нижней части рабочего пространства). Эта технология позволяет использовать информацию хранилища одновременно с созданием графических элементов, без необходимости открывать отдельные окна хранилища, как это делается в Rational Rose.

В среде визуального моделирования инструментального средства Poseidon видимость, понятность и возможность манипулирования диаграммой облегчается использованием разных цветов и окна обзора, чтобы оценить размер всей диаграммы. Окно обзора – характерная помощь при работе с большими моделями.

5.5.8. Интегрированные среды разработки

Среды программирования обычно называются **интегрированными средами разработки** (integrated development environments – IDE). IDE автоматизируют многие утомительные процессы программирования и позволяют разработчикам сосредоточиться на более важных проблемах. Стандартные возможности IDE включают завершение кодирования, поддержку стандартных приемов программирования (ревизия кода), интеграцию с инструментальными средствами конструирования типа Ant or make и т. д. Этот раздел, подобно ос-

тальной части книги, обсуждает среды программирования на основе Java. Однако многие выводы применимы также и к другим средам.

Хотя разные IDE и схожи в стандартных особенностях, все-таки имеются различия между ними. Некоторые различия касаются интеграции IDE с другими инструментальными средствами программной инженерии, в особенности, с инструментальными средствами моделирования) и с инструментальными средствами, поддерживающими коллективную работу (инструментальные средства управления изменениями и конфигурацией). Сказывается также, что различные IDE-продавцы нацелены на различные рынки разработчиков. Разные IDE могут различаться следующими особенностями:

– интеграция с моделированием ПО;

– поддержка возможностей и характера прикладной разработки;

– интеграция с распределенными средами разработки и с инструментальными средствами управления изменениями и конфигурацией.

Так как UML становится стандартным языком моделирования, его поддержка и пригодность для IDE становится важной. UML-проекты обеспечивают программистов абстракциями, необходимыми, чтобы рассматривать ПО в процессе разработки с различных и более понятных ракурсов.

5.6. Планирование и отслеживание проекта программного обеспечения

В данном разделе рассмотрено планирование и отслеживание проекта ПО. Это составляет содержание процесса управления в рассмотренном ранее пятиугольнике разработки ПО. Различие между планированием проекта и управлением процессом довольно неопределенно. Планирование и отслеживание проекта ПО является непрерывной операцией оценки того, сколько времени, денег, трудозатрат и ресурсов должно быть израсходовано на проект.

Будучи не однократной, а непрерывной операцией, планирование проекта ПО можно считать всеобъемлющей операцией управления продуктами и процессами разработки ПО. В частности, планирование проекта ПО включает не только само планирование, составление бюджета и отслеживание, по также и анализ рисков, гарантию качества, управление людскими ресурсами и управление конфигурацией. Однако в данном разделе считается, что планирование эквивалентно собственно планированию и отслеживанию проекта.

Интересно, что сопутствующие проблемы анализа рисков, гарантии качества, управления конфигурацией и т. д. имеют многопро-

ектный смысл. Они актуальны и в случае наличия нескольких проектов. Это – другая причина отделить планирование для единственного проекта от многопроектных аспектов.

Планирование проекта ПО начинается там, где завершается стратегическое планирование и бизнес-моделирование. Стратегическое планирование и бизнес-моделирование решают вопросы, связанные с организационными задачами, показателями и целями.

5.6.1. Разработка плана проекта

Как сказано выше**, планирование проекта** – не отдельная стадия жизненного цикла разработки. Это – всеобъемлющая операция, охватывающая все стадии жизненного цикла. Хотя некоторые модели жизненного цикла, например. RUP (раздел 1.3.2), рассматривают планирование как самую первую операцию в начале проекта, они поясняют, что это – только начальное планирование

Никакие разумные оценки стоимости (бюджета) и времени (планы) невозможны, пока не будут определены, по крайней мере, требования пользователя.

Исследование Бёма и др. указывает, что первоначальные плановые оценки после изучения выполненного проекта могут измениться в одну или другую сторону в четыре раза по сравнению с реальной стоимостью/ временем. Затраченными на выполнение проекта. Если реальные израсходованные стоимость/ время равны некоторой величине x, то первоначальные оценки могли бы быть в диапазоне от $0{,}25x$ до $4x$. Последовательные коррекции плана оценивают все точнее реальное значение x.

Оценка стоимости/ времени (бюджет/ план) – только один аспект планирования проекта ПО. Практически документ бюджета/ плана может быть одним из нескольких документов планирования. Другие документы могут включать план качества, план тестирования, план использования людских ресурсов, план управления конфигурацией и т. д. Они могут быть отдельными документами или частями общего планового документа проекта. Обычно рекомендуются следующие разделы типичного плана проекта:

Введение – определить проектные показатели, основную стоимость и ограничения по времени.

Организация проекта – описать организацию коллектива разработчиков.

Анализ рисков – определить риски и способы управления ими.

Требования к ресурсам аппаратного и программного обеспечения – определить аппаратное/ программное обеспечение, необходимые для разработки.

Выполнение – определить проектные показатели, контрольные точки и подлежащие сдаче продукты.

План проектных работ – определить распределение времени и ресурсов по операциям.

Механизмы контроля и передачи сообщений – определить потребности в механизмах контроля и управляющих сообщениях.

Шаги в разработке плана проекта ПО подобны тем, которые требуются в других видах творческих проектов (рис. 5.4).

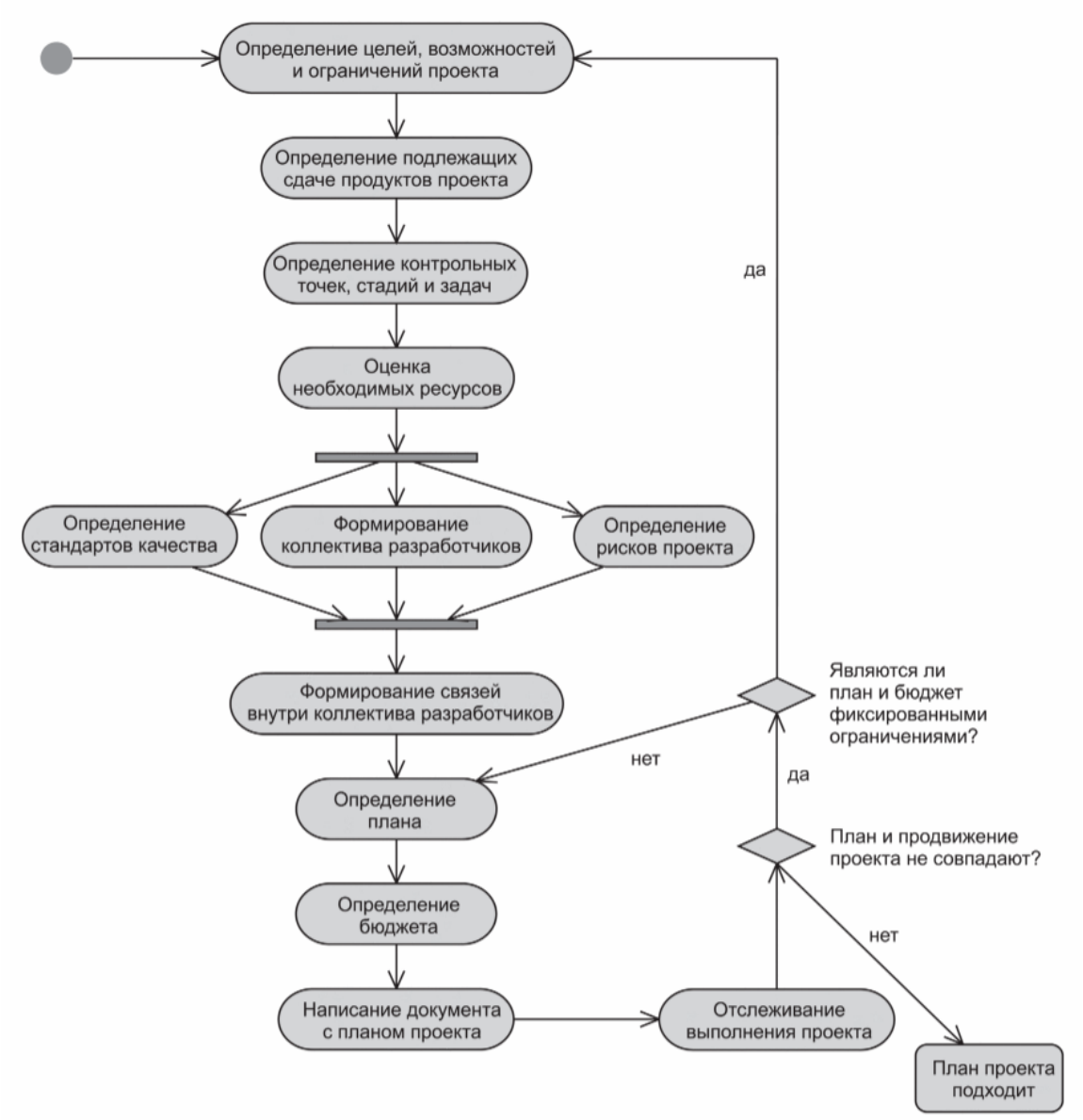


Рис. 5.4. UML-диаграмма операций, которая иллюстрирует типичную последовательность шагов

Планирование проекта начинается с определения *показателей, возможностей и ограничений* проекта. На практике показатели и возможности проекта будут определены до начала планирования проекта (поскольку они являются составными частями стратегического планирования и бизнес-моделирования). Два типичных ограничения проекта – время и деньги, так как проекты имеют сроки и ограниченные бюджеты. Другие ограничения могут касаться пригодности ресурсов, в частности, людских ресурсов.

Подлежащие сдаче продукты проекта – это программные продукты или сервисы, которые удовлетворяют проектным показателям. Они поставляются заинтересованным сторонам проекта (владельцам проекта и пользователям). Эти продукты должны удовлетворять требованиям пользователя, соответствовать проектной спецификации и удовлетворять ряду стандартов качества. В современных методах итеративных и пошаговых разработок проект, вероятно, будет иметь несколько подлежащих сдаче продуктов.

Как только подлежащие сдаче продукты станут известны, в работе можно выбрать контрольные точки, стадии и задачи. Иногда это называется **структурной декомпозицией работы. Контрольная точка** – существенное событие в проекте. Она отмечает конец некоторой операции процесса и используется, чтобы контролировать продвижение в проектировании. Контрольная точка обычно используется, чтобы зафиксировать завершение создания подлежащего сдаче продукта, но она может зафиксировать и другие виды результатов работы над проектом, например, просто завершение важного шага проектирования. Контрольная точка может также показать завершение проектной стадии (а подлежащий сдаче продукт обычно является результатом некоторой заключительной стадии). Проекты (и стадии) состоят из **задач,** которые являются **операциями** с ясными началом и концом.

Планирование первых шагов проектирования обращает внимание на то, *что* это за проект.

Оценка потребностей ресурсов – из области *как* выполнить проект – необходима для решения, как выполнить проектные задачи. **Ресурсы** – это люди, оборудование, материалы, поставки, программное и аппаратное обеспечение и т. д., необходимые для решения проектных задач. Оценка потребностей ресурсов может быть только начальным шагом в целом ряду операций планирования, таких, например, как планирование комплектации штата.

Формирование *коллектива разработчиков проекта* – это операция, которая управляет потребностями людских ресурсов. Одно дело – знать, какие людские ресурсы необходимы, а вот формирование коллектива – совсем другой вопрос. Некоторые члены коллектива могут быть взяты непосредственно в организацию, другие должны быть наняты на полный рабочий день или как работающие по контракту. Создание хорошего коллектива с дополняющими навыками, наличием духа товарищества, без личных напряженных отношений и заинтересованного в проекте – одна из наиболее серьезных проблем менеджера проекта.

Планирование проекта включает определение стандартов качества.

Качество – относительное понятие. Оно на виду у всех. Изделие высокого качества – это изделие, которое удовлетворяет пользователя. Это означает программный продукт, который был поставлен вовремя и в пределах выделенного бюджета, реализует требования пользователя, имеет дружественный интерфейс, не отказывает в работе и т. д. Более высокое качество может означать более высокую стоимость. Определение стандартов качества должно принимать во внимание воздействие политики качества на продолжительность проектирования и закладываемый бюджет и даже на возможности проекта.

Планирование проекта включает также определение проектных рисков.

Риски – неблагоприятные события, которые часто трудно ожидать, и даже когда они определены и известны, еще неясно произойдут они или нет. На работнике, занимающемся планированием проекта, лежит ответственность за определение стратегии *предотвращения рысков* и выработку *планов на случаи непредвиденных обстоятельств*, если риск произойдет.

Установление связей в коллективе разработчиков – важная задача в любых, но особенно в малых проектах. В малых проектах члены коллектива могут связываться через неофициальные встречи, обмены по электронной почте и т. д. В больших проектах связи внутри коллектива должны быть запланированы и обеспечены технологией. Технологические решения включают формирование рабочей группы, управление временем, управление конфигурацией и связанные с этим системы ПО, способные обеспечивать обмен мультимедийными документами.

Шаг определения **плана** проекта – нечто, связанное с оценкой времени, необходимого не только для самой разработки, но также и

для управления проектом, гарантии качества изделий и управления рисками. Планирование включает также и распределение ресурсов по задачам.

Шаг определения **бюджета** проекта – нечто вроде оценки стоимости ресурсов, которые нужны для задач. Имеются различные технологии такой оценки. Ни одна из них не совершенна и, по крайней мере, пара их должна использоваться для проекта в целях проверки. В некоторых случаях затраты могут быть рассчитаны на основе плана, используя тарифные ставки и систему оплаты для располагаемых людских ресурсов, систему затрат на задачи и т. д. В других случаях затраты могут быть получены на основе известных метрик затрат подобных проектов. Можно также оценить бюджет, используя алгоритмические модели, которые предсказывают стоимость, основанную на проектных характеристиках (параметрах), заложенных в математическую модель.

Написание **документа с планом** проекта – кульминация ранее выполненных шагов планирования. Этот документ должен быть передан работникам управления и членам коллектива проектировщиков. Передача может быть в электронной форме с помощью электронной почты, или путем размещения документа на Web-сайте Intranet.

План проекта регулярно пересматривается в процессе **отслеживания выполнения проекта** в случае выявления несовпадения с планом.

5.6.2. Оценка бюджета проекта

Оценка бюджета проекта – тернистая проблема. Имеется много предложенных подходов к оценке затрат, но только пара из них имеет научное обоснование. Кроме того, научные подходы дорогостоящи. Имеется компромисс между трудозатратами, необходимыми для достижения более точного и достоверного бюджета, и величинами, которые этот бюджет обеспечивает.

Как установлено, трудозатраты по формированию бюджета быстро растут с уровнем детализации, рассматриваемой при определении факторов, которые влияют на вычисления. Точность бюджета, по-видимому, устойчиво улучшается с ростом детализации. Удобство сопровождения бюджета, однако, является высоким для упрощенных.

Реальная сметная стоимость сама по себе является компромиссом между точностью и удобством сопровождения. До некоторой точки совокупный эффект этих трех факторов улучшает сметную

стоимость. Однако на некотором уровне детализации фактическая сметная стоимость начинает уменьшаться.

Экспертная оценка – метод, в котором оценки запрашивают от различных экспертов, а затем они обсуждаются и выверяются, чтобы достичь согласованного результата. **Оценка по аналогии** может использоваться, когда имеются подобные законченные проекты в той же самой предметной области и фактические их затраты известны. Бюджет проекта тогда оценивается по аналогии с тем, что произошло в прошлом.

Более сложные методы оценки бюджета могут быть поделены на две большие группы. Первая группа состоит из восходящих методов, которые рассматривают задачи и ресурсы, размещенные в подготовленном графике проектных работ ранее, и получают бюджет, суммируя затраты на ресурсы плюс другие фиксированные расходы. Эти методы обсуждаются ниже под названием «оценка бюджета на основе графика выполнения».

Второй надежный подход к оценке бюджета известен как «алгоритмическая оценка бюджета». В этом подходе используется для получения затрат некоторая оцененная метрика размера ПО, типа числа строк кода или числа функциональных единиц. Затраты получают, задавая в математическую модель значения метрик и большою числа параметров настройки. Сама модель

Оценка бюджета на основе графика выполнения

Эта оценка предполагает, что существует план исходных данных с ресурсами, распределенными по задачам. Он называется планом на основе **исходных данных** или просто планом. Поскольку планы имеют тенденцию изменяться, инструментальное средство управления проектами может обслуживать различные исходные данные и формировать отдельные оценки бюджета для каждого набора исходных данных.

Теоретически кажется, что оценку бюджета на основе графика выполнения легко произвести. Так как ресурсы в исходных данных назначены задачам, все, что остается сделать, это назначить стоимости затрат для ресурсов, добавить накладные расходы и другие фиксированные затраты и воспользоваться инструментальным средством управления проектами, чтобы вычислить бюджет для проекта. Дьявол запрятан в деталях. Например, ресурсы могут накапливать затраты в различные моменты времени в процессе выполнения задачи. Нормы и фиксированные затраты могут быть разными в различные периоды

времени. Имеются некоторые узаконенные правила, влияющие на ставки заработной платы и затраты. На вычисления могут влиять принципы бухгалтерского учета и т. д. Однако первоначальное вычисление бюджета тривиально по сравнению с отслеживанием бюджета в ответ на изменения плана, изменения исходных данных и других проектных условий.

Как сказано, оценка бюджета требует ввода **ставок** и других затрат для рабочих и материальных ресурсов. Кроме **стандартных ставок** могут быть ставки за сверхурочное время и ставки за использование. **Ставки за сверхурочную работу** оплачиваются из рабочего ресурса, если ресурс используется за пределами обычных рабочих часов. Сверхурочная работа – не дополнительная работа для задачи, но она может использоваться, чтобы сократить продолжительность ее выполнения. **Ставки за использование** фактически не являются ставками, а совокупностью затрат, связанных с использованием ресурса. Они могут использоваться как замена других ставок или в дополнение к ним.

Методы наращивания затрат должны быть определены для ресурсов наряду со ставками и другими затратами. Наращивание можно выполнять с начала задачи, с конца задачи или пропорционально (как выполняется работа). Пропорциональный метод наиболее часто используется и накапливает затраты в зависимости от процента завершения задачи. Методы наращивания применяются как к гибким затратам, так и к фиксированным затратам.

Как только затраты ресурсов будут известны, можно вычислить общие стоимости задач. Обратите внимание, что затраты на задачи могут включать фиксированные расходы, назначенные скорее на задачу, чем на ресурс, использованный задачей. Например, фиксированные затраты могут быть в отношении расходных материалов и подобных расходов.

Инструментальное средство управления проектами может также обслуживать распределение затрат между ресурсами. Как и раньше, распределение затрат ресурсов все же не является совершенным.

Алгоритмическая оценка бюджета

Ну, хорошо, допустим, что оценка бюджета на основе графика выполнения дает приемлемые значения при разумных трудозатратах. Однако на практике оценка бюджета не должна полагаться только на одну технологию оценки. Точность оценки бюджета должна быть проверена путем применения другой технологии. Проблема напоми-

нает использование различных цен для сервисов, предоставляемых продавцами: единая цена может оказаться недостаточной для конкретного случая.

Алгоритмическая оценка бюджета вторая лучшая технология оценки после оценки на основе графика выполнения. Она использует некоторую меру (метрику) масштаба задачи и применяет к ней полученный опытным путем алгоритм, чтобы определить бюджет, необходимый для решения задачи. В производственных отраслях типа выпуска автомобилей проблема проста. Учитывая время и стоимость создания одного автомобиля, можно получить время и стоимость всего производственного «проекта».

В технических дисциплинах, и в особенности в разработке ПО, проблема намного хитрее. Для начала – технический проект производит единственное уникальное изделие, а не большое количество одного и того же продукта. Даже если может быть получена метрика, которая позволяет разложить изделие на несколько компонентов, эти компоненты совершенно разные. ПО простым образом не масштабируется. Каждый компонент требует различных трудозатрат, даже если он столь же примитивен, как строка кода или функциональная единица. Большинство затрат относится к рабочим ресурсам, и людским ресурсам, в частности, но не к материальным ресурсам. Наконец, так как каждое изделие уникально, трудно заранее предсказать размер потребных ресурсов.

Функциональные единицы дают более высокий уровень декомпозиции кода, чем строки кода (то есть, они обеспечивают менее детальную калибровочную информацию). Функциональные единицы – не модули ПО или компоненты, как можно предполагать по их имени. В наиболее частом использовании функциональные единицы вычисляют, анализируя ПО с точки зрения пяти особенностей (входы, выходы, файлы данных, запросы и внешние интерфейсы), определяя их число, одновременно корректируя вычисления в зависимости от представляемой сложности этих особенностей.

Объектные единицы находятся даже на еще более высоком уровне декомпозиции кода, чем функциональные единицы. Подобно функциональным единицам их получают, используя вычисленное количество различных особенностей ПО. Особенностями здесь являются экраны пользовательского интерфейса, сообщения ПО и компоненты ПО. Как и с функциональными единицами, вычисление корректируют, используя весовые коэффициенты, отражающие сложность особенностей. Вычисление также корректируется для удаления

многократно используемых объектных единиц. Это приводит к мере, называемой новыми объектными единицами (new object point – NOP).

При наличии калибровочной информации алгоритмическая модель оценки затрат определяется формулой следующего вида:

$$effort = c \cdot size^{k} \qquad (5.1)$$

где *effort* – трудозатраты; *size* – размер ПО.

Константы *c* и *k* устанавливаются опытным путем и задаются самой моделью. Показатель *к* отражает сложность задачи и непропорциональное увеличение трудозатрат в больших проектах. Этот показатель обычно находится в диапазоне от 1 до 1,5. Множитель *c* – другой коэффициент вычисления проектной сложности, основанной на оценках атрибутов типа требуемой надежности изделия, навыков коллектива, доступных инструментальных средств ПО, и т. д.

Наиболее известные алгоритмические модели – так называемые COCOMO-модели. Аббревиатура COCOMO означает COnstructive COst MOdel – конструктивная модель затрат. Имеются две COCOMO-модели, обе разработаны под руководством Барри Боэма. Первоначальная модель, разработанная в 1980-х гг., теперь известна как C'OCOMO 81. Улучшенная версия, опубликованная в 2000 г., известна как COCOMO II.

COCOMO 81

Несмотря на то. что COCOMO II эффективней COCOMO 81 почти для всех традиционных проектов ПО, использующих языки третьего поколения типа КОБОЛ или С, с методической точки зрения целесообразно объяснить принципы COCOMO 81 перед рассмотрением COCOMO II**.**

Фактически COCOMO 81 представляет множество трех подходов к оценке затрат. Эти подходы используются, соответственно, для небольших, среднего размера и сложных систем. Каждый подход имеет базовую формулу оценки, основанную на предсказанном размере кода. Имеется ряд формул, позволяющих выбрать из них нужную для каждого подхода на основе характеристик проекта. Эти три способа и связанные с ними базовые формулы, следующие:

Базовый (или основной) COCOMO:

$$effort = 3{,}2 \cdot size^{1{,}05}. \qquad (5.2)$$

Промежуточный (или полу-изолированный) COCOMO – этот способ включает в оценку ряд коэффициентов, определяющих вели-

чину затрат (или коэффициентов затрат), чтобы масштабировать трудозатраты номинальной разработки. Эти коэффициенты основаны на оценках изделия, аппаратного обеспечения, персонала и атрибутов проекта:

$$effort = 3{,}0 \cdot size^{1{,}12}. \quad (5.3)$$

Продвинутый (или вложенный) COCOMO – этот способ получен из промежуточной версии, но коэффициенты затрат оценивают воздействия на каждой стадии жизненного цикла разработки:

$$effort = 2{,}8 \cdot size^{1{,}20}. \quad (5.4)$$

Например, используя базовый COCOMO и предполагая, что система будет состоять из 100000 строк кода (*size* = 100 KLOC), оцененные трудозатраты, выраженные в человеко-месяцах:

$$effort = 3{,}2 \cdot size^{1{,}05} = 3{,}2 \cdot 125{,}9 =$$
$$= 403 \text{ person-months (человеко-месяцев)}$$

В оценке бюджета на основе графика выполнения план используется, чтобы получить бюджет для всего проекта, а также частные бюджеты для отдельных задач и ресурсов. Алгоритмические методы не связывают с планом, за исключением оценки времени разработки, соответствующего полученным трудозатратам (*effort*). Эмпирические формулы оценки полного времени разработки для трех подходов COCOMO будут соответственно [9]:

простой COCOMO*:*

$$time = 2{,}5 * effort^{\,0{,}38}; \quad (5.5)$$

промежуточный COCOMO:

$$time = 2{,}5 * effort^{\,0{,}35}; \quad (5.6)$$

продвинутый COCOMO:

$$time = 2{,}5 * effort^{\,0{,}32}. \quad (5.7)$$

Ясно, что COCOMO не столь прост, как взятие размера ПО и использование его в формуле, чтобы оценить трудозатраты. Точность оценки в первую очередь определяется точностью коэффициентов затрат и других масштабирующих коэффициентов. Это в свою очередь зависит от пригодности метрик из других проектов. Организация,

разрабатывающая ПО, должна знать свою историю (и иметь ее оценку в числах), чтобы планировать свое будущее.

COCOMO 81 позволил накопить большой объем показателей прошлых лет от множества организаций в надежде, что опыт других организаций может использоваться там, где локальные метрики непригодны. Это было полезно во времена, когда КОБОЛ использовался как универсальный язык программирования бизнес-систем, а приложения не были столь разнообразны, как сегодня. Никакой такой исторической информации нет для современных проектов. Каждая организация должна собрать метрики от предыдущих проектов, чтобы иметь хоть какую-то надежду уверенно планировать в будущем.

COCOMO II

COCOMO II основан на тех же принципах, что и COCOMO 81, но он учитывает пару важных недостатков COCOMO 81. Модель была калибрована по результатам анализа опытных данных, собранных (первоначально) из 83 проектов ПО. Подобно всем алгоритмическим моделям, COCOMO II требует калибровочной информации. Кроме LOC и FP COCOMO И вводит объектные единицы (object point – OP) как вариант калибровки. Подобно COCOMO 81 COCOMO II использует разнообразные коэффициенты затрат, масштабирующие коэффициенты и другие процедуры настройки.

COCOMO II существенно более сложен, чем COCOMO 81, что отражает:

– возросшую сложность ПО и предметных областей;

– более широкое разнообразие моделей жизненного цикла и технологий разработки ПО;

– пропорциональный рост генерации и повторного использования ПО в сравнении с ручным программированием;

– непрерывный характер планирования и оценки затрат.

В отличие от COCOMO 81, который обеспечивал оценки для жизненного цикла типа каскада и процедурного программирования, COCOMO II приспособлен к итеративным жизненным циклам и объектно-ориентированному программированию. Он признает использование опытных образцов, шаги, короткие циклы, использование генераторов программ, разработку с использованием объединения компонентов и т. д.

Более подробное изложение методики COCOMO II не приводится из-за ограниченного объема данного пособия.

5.6.3. Управление персоналом, рисками, качеством, изменениями

Организации, разрабатывающие ПО, постоянно стремятся улучшать методику и практику своих разработок. Требуется также, чтобы в организации знали об *основных факторах*, к которым приведет усовершенствование методики и практики процесса в этой организации. Для оценки уровня компетенции коллектива разработчиков программных продуктов вводится понятие ***уровня зрелости*** управления (Management Maturity Levels) – это этапы развития организации в соответствии со стандартизованными моделями оценки уровня зрелости управления. Универсальной моделью оценки уровня зрелости считается Capability Maturity Model Integrated (далее CMMI). CMMI описывает шкалу из пяти уровней зрелости, основанных на том, насколько последовательна компания или организация в следовании общим повторяющимся процессам при выполнении своей работы. На рисунке 33 представлены уровни зрелости модели CMMI. Нижний уровень шкалы описывает компании без повторяющихся процессов, где большая часть работы хаотична и сумбурна. Верхний уровень описывает компании, которые используют определенные и повторяющиеся процессы, собирают метрики для непрерывного улучшения своих процессов, а также на регулярной основе ищут творческие методы, позволяющие работать лучше.

Международная организация по стандартизации ISO (International Standard Organization) применяет CMMI для создания международных стандартов. В рамках данной работы мы будем обращаться к стандарту ISO 9004–2010 «Менеджмент для достижения устойчивого успеха организации». Согласно ISO 9004–2010, устойчивый успех организации достигается за счет ее способности отвечать потребностям и ожиданиям своих потребителей и других заинтересованных сторон на долговременной основе и сбалансированным образом. Устойчивого успеха можно добиться посредством эффективного менеджмента организации, путем осознания организацией среды своего существования, за счет обучения и должного применения улучшений и (или) инноваций. Данный стандарт предлагает методику самооценки уровней зрелости на основе опроса, по результатам которого можно сделать вывод об уровне зрелости организации.

Как видно на рис. 5.5, CMM-модель определяет пять возрастающих уровней зрелости процесса: 1) начальный (initial), 2) повторяемости (repeatable), 3) регламентируемости (defined), 4) управляемости (managed) и 5) оптимизируемое (optimizing). CMM стала

эффективным стандартом качества процесса. Модель имеет связанный с ней механизм аккредитации, который позволяет организациям подвергнуться процедуре ревизии, которая засвидетельствует зрелость их процесса. Многие контракты на разработку ПО требуют доказательства конкретного минимального уровня зрелости процесса от поставщиков ПО и подрядчиков.

Рис. 5.5. Модель оценки уровней зрелости организации
Capability Maturity Model Integrated (CMMI)

На *начальном уровне* зрелости организация на самом деле не имеет никакого предсказуемого процесса. Любой успешный проект – вопрос удачи, а не результат управляемого процесса. Очевидно, результат хорошей работы определяется некоторыми основными индивидуумами. Если эти индивидуумы оставляют организацию или когда возникает некоторая другая кризисная ситуация, нет никаких установленных порядков, которым нужно следовать, чтобы вернуть проект в нужное русло.

Переход на *уровень повторяемости* требует усовершенствования дисциплины процесса. Дисциплина – результат планирования и отслеживания операций, чему до некоторой степени способствуют гарантии качества и управление конфигурацией. Процесс на этом уровне все еще более интуитивен, нежели запланирован, но опыт

планирования и отслеживания предыдущих проектов может быть успешно использован (повторен) в текущем проекте Переход на *уровень регламеитируемости* требует улучшения в определении процесса. Основные факторы в достижении уровня регламентируемости – управление людскими ресурсами и рисками. Определяющие процессы занимаются рисками и кризисными ситуациями. Эти процессы облегчаются широким использованием инструментальных средств разработки ПО.

Переход на *уровень управляемости* требует улучшения управления производственным процессом. На этом уровне тщательно отбираются процессы и метрики продуктов, чтобы измерить качество ПО и производительность людей. Метрики позволяют на ранних этапах определить проблемы и предпринять обходные или корректирующие действия.

Переход на *уровень оптимизируемости* требует улучшения самой области совершенствования процесса. Основной фактор в совершенствовании процесса – управление изменениями и конфигурацией, включая управление дефектами и повышением темпов роста, а также непосредственными изменениями в технологии и процессе. Должна быть запланирована и субсидирована целая область усовершенствований процесса.

Управление людьми – это не то, что обычно предполагается под этим термином. Связано это с рядом причин:

1. Технология изменяет все аспекты бизнеса.

2. Изменение бизнес-среды влияет на организационные структуры, приводя к формированию сетевых организаций, не жестко связанных сотрудничающих единиц.

3. Новые организационные структуры подразумевают новые стили управления.

4. Новые стили управления должны быть допускающими риски и творческими.

5. Допущение риска и творчество не сочетаются с бюрократическим управлением людьми, основанным на ответственности и методиках.

6. Сильной стороной современных рабочих коллективов является гибкое сотрудничество людей, где отдельные индивидуумы не обязательно являются лояльными к организации, но полагаются на рабочую среду и ряд абстрактных, не связанных с работой потребностей.

Современные организации имеют тенденцию выглядеть как *сетевые структуры* не жестко связанных единиц, объединенных ско-

рее коллективной общностью, нежели собственностью. Люди в **сетевой организации** обладают уникальными представлениями о мотивации, лояльности, организационном воздействии, лидерстве, достижениях, удовлетворении, вознаграждении и т. д. Старый стиль иерархического управления не применим для коллективов в сетевой организации. Коллективы разработчиков ПО находятся на переднем крае этой волны изменений.

Привлечение и мотивация людей

Привлечение и мотивация людей является сущностью процесса создания коллектива. Коллектив формируется на основе плана проекта и связанного с ним **плана управления комплектацией персонала.** Реализация формирования коллектива включает подбор менеджеров проектов, руководителей проектов и всех других членов коллектива. В зависимости от ситуации членами коллектива могут быть текущие служащие, назначенные на проект, или может потребоваться сформировать коллектив, нанимая новых работников или привлекая внешних подрядчиков и консультантов. Современные сетевые организации одобряют **самоуправляемые коллективы,** но хорошее управление и лидерство – необходимые условия для успеха любого проекта. ем больше проект, тем это утверждение более верно. Подбор хорошего менеджера проекта, который знает, как и когда управлять, как и когда руководить, является критическим решением владельца проекта.

Обычно задается следующий набор различных характеристик *руководителей проектов* и *менеджеров*.

Руководители должны уметь убедить других делать то, что нужно, организовать и сплотить их вокруг своего видения предмета. Хорошие руководители подбирают членов коллектива, которые верят в их видение. Руководители задают направление и временные границы и имеют способность привлечь таланты для работы. Менеджеры обычно ориентированы на задачи, связанные с понятиями типа планов, управления, бюджетов, стратегий и методик. Они – универсалы с широкой базой планирования и организационных навыков, и их первичная цель – удовлетворять потребности участников разработки. Они также обладают мотивационными навыками и способностью распознавать и вознаграждать поведение.

Важный момент – это то, что менеджер проекта должен также быть и руководителем, в то время как обратное не обязательно. Хороший менеджер должен и управлять, и вдохновлять. Руководство

особенно эффективно на больших проектах. В таких проектах расстояние между менеджерами и исполнителями решений может включать несколько уровней управления. Чтобы выполнить то, что надо, менеджеры должны доверять членам коллектива, а члены коллектива должны быть как вдохновлены, так и заинтересованы.

Формирование коллектива

Члены коллектива привлекаются путем назначения служащих на проект и/ или найма новых работников. *Перевод служащих* между проектами является прерогативой высшего руководства. Менеджер проекта должен пытаться найти подходящих, потенциально доступных и заинтересованных служащих и запросить их перевод. Подходящие не обязательно означает лучшие служащие. Лучшие люди находятся в большом спросе. Переназначение их означает, что пострадают другие проекты. Наличие лучших людей повышает надежды на удачное управление проектом. Наконец, лучшие люди могут и не вписаться в коллектив и могут вызвать нежелательные напряженные отношения.

Наем новых служащих дает лучшую возможность нахождения людей, которые могут обеспечить определенные потребности проекта и коллектива, по это всегда связано с большим риском неправильной оценки претендентов. Решение, кого нанимать, принимается с учетом трех видов входной информации:

– информация, представляемая претендентами, об их портфеле занятости;

– рекомендации, полученные на претендентов от их прошлых работодателей и других внешних поручителей;

– результаты интервью и любых профессиональных испытаний, которым были подвергнуты претенденты.

Полное использование всех трех видов информации обязательно, пока нет другого «вызывающего доверие» вида информации от людей, которые знают претендента. Хотя такой вариант часто «для протокола», он, конечно, играет роль в практике найма.

Испытания с целью оценки претендентов делятся на две группы: 1) испытания способностей и 2) психометрические испытания. **Испытания способностей** предназначены раскрыть навыки человека выполнять некоторые задачи типа программирования на конкретном языке. **Психометрические испытания** предназначены раскрыть позицию человека и пригодность для некоторых типов задач, типа способности принимать решения. Испытания обычно проводятся в огра-

ниченное время и с другими экзаменационными воздействиями. Они не могут быть истинным отражением способности человека решать задачи в нормальной рабочей среде.

Люди привлекаются, чтобы заполнить определенные потребности проекта и коллектива. Эти потребности разнообразны и дополняют друг друга. Типичные факторы, которые определяют выбор штата, следующие:

– предметная область и опыт в бизнес-процессе;

– технические знания, опыт и знакомство с методами и инструментальными средствами:

– образовательная база;

– способность решать задачи;

– коммуникабельность;

– адаптируемость, готовность и пригодность;

– позиция, амбиции и инициатива;

– индивидуальность.

Как сказано мимоходом, сильной стороной современных рабочих коллективов является гибкое сотрудничество людей, где отдельные индивидуумы не являются обязательно лояльными к организации, но полагаются на рабочую среду и ряд абстрактных, не связанных с работой потребностей. Привлечение служащих, вероятно, будет меньшим беспокойством в управлении, чем их мотивация и сохранение. Как известно, инженеры ПО обычно бывают богатыми в денежном плане и бедными в смысле свободного времени. Мотивация и сохранение таких служащих требует нетрадиционных подходов типа разрешения работы на дому или предоставления большего количества свободного времени.

Теории мотивации

Мотивация может быть внешней или внутренней.

Внешние факторы мотивации – это материальные блага, которые могут включать премии, использование служебной машины, фондовые опционы, дарственные, возможности обучения и т. д. Внутренние факторы мотивации различны для разных индивидуумов. Некоторые люди по своей природе стремятся получить результат – это часть их характера. Культурные и религиозные влияния также являются внутренними факторами мотивации. Награды и признания являются примерами внешних факторов мотивации. Имеется много мотивационных теорий, которые пробуют определить, что побуждает людей в их действиях и какие факторы влияют на профессиональное

поведение людей. Явно или неявно, все популярные теории подтверждают, что люди мотивированы лучше всего, когда дана возможность реализовать абстрактные потребности и цели высокого уровня для «души». Фундаментальные потребности и цели низкого уровня для «тела» – получение материальных благ типа пищи, одежды и даже денег являются слабыми факторами мотивации в добавление к «стандартному уровню существования». Стандартный уровень существования означает здесь, что эти факторы мотивации предоставляются человеку только в необходимом количестве. Увеличение этого количества для лучшей мотивации людей приносит только краткосрочные эффекты.

Организация связи в проекте

Связь – это операция обмена информацией и знаниями. Это процесс передачи сообщений от отправителей к получателям. Для многих задач и ролей выполняемой работы возможности связи могут быть более важны, чем технические возможности. Возможно, это истинно и для работ по управлению. Каждый аспект творческой деятельности, типа проектов разработки ПО, обеспечивается связью.

Управление процессом создания ПО рассматривает связь проекта, по крайней мере, с четырех точек зрения:

– формы связи;

– линии связи;

– облегчающие (тормозящие) факторы связи;

– связь в разрешении конфликтов.

Формы связи

Сообщения передаются в некоторой закодированной форме. Код может быть устным, представленным в тексте, в рисунках, в символах, переданным выражением лица или интонацией и т. д. Получатель должен понять код отправителя, чтобы связаться с ним. Сообщение может быть конфиденциальным, публичным, формальным, неофициальным, внутренним, внешним, горизонтальным (между равными по положению), вертикальным (между руководителями и подчиненными) и т. д.

Типичные **формы связи:**

– случайные связи по принципу «от человека к человеку»;

– встречи в помещении по принципу «от человека к человеку»;

– телефонные беседы по принципу «от человека к человеку»;

– конференции в Интернете по принципу «от человека к человеку» (текстовые, голосовые и/ или видео);

– телефонный автоответчик;
– факсимильная почта;
– почта;
– курьерская почта;
– Web-страницы.

Различные формы связи имеют свои уникальные преимущества и недостатки. Формы *«от человека к человеку»* могут передавать значительно большее количество информации изменением тона, выражениями лица, движениями тела и т. д. Если не осуществлять запись, связь «от человека к человеку» может передавать даже большее количество информации, допуская большую степень свободы в выражениях без опасения последствий от чрезмерного распространения записей текста/ голоса/ видео.

Электронная почти (E-mail) – мощная, но опасная среда связи. Мало того, что сообщения электронной почты могут быть распространены далее, они могут также быть отправлены или скопированы по ошибке. Электронная почта дает вводящее в заблуждение чувство конфиденциальности связи «от человека к человеку», но это может закончиться формальным и широким распространением информации. Кроме того, при передаче намного более вероятно, чем у других форм связи «от человека к человеку», что сообщение, первоначально обеспечивающее конфиденциальность, затем становится публичным. В отличие от *обычной почты* отправитель электронной почты, как правило, может знать, дошло ли сообщение до получателя или даже прочитано ли им. Вообще, электронная почта не должна использоваться в щекотливых ситуациях.

Web-страницы – мощная среда связи. Эти страницы обеспечивают быструю передачу информации большим группам людей. Они не разрушительны в том смысле, что адресуемые получатели могут посещать страницы по своему желанию. Однако связь через Web-страницы должна гарантировать своевременное обновление информации в них, иначе они будут восприниматься как ненадежные, и получатели перестанут к ним обращаться.

Из всех *почтовых* форм текстовая форма доминирует в формальной связи. Голосовые и видео-формы сравнительно нечасты. Они требуют слишком большого количества трудозатрат и времени для «расшифровки», чтобы играть существенную роль в формальных обменах информацией.

Показатели связи

На практике обычно не требуется, чтобы каждая конкретная часть информации была передана всем участникам проекта. И слишком большая, и слишком малая связь может вредить проекту. Члены коллектива должны быть информированы относительно всего, что воздействует или касается задач, которые они выполняют. Участники проекта должны также быть информированы относительно глобальных проблем, проектных направлений, отобранных политических решений и т. д., чтобы они чувствовали себя неотъемлемой частью проекта и организации.

Опасность иерархических структур и вертикальных сообщений состоит в том, что люди на одном и том же уровне имеют недостаточную связь. Имеются различные пути улучшения связи внутри таких коллективов. Один из них – это такой подбор членов коллектива, чтобы гарантировать надлежащее объединение экстравертов и интровертов. **Экстраверт** – человек, который легко вступает в связь с другими людьми. **Интроверт,** в противоположность этому, застенчивый, тихий человек, часто очень хороший слушатель. (Кстати, навыки внимательно слушать столь же важны в связи, как и способность говорить.)

Выше было сказано, что значительное количество связей является несущественным. Это и истинно, и желательно. Организация рабочего места может существенно облегчить или запретить заранее спланированную и/ или **несущественную связь**. Например, работа в помещениях с открытой планировкой с или без индивидуальных кабин и поощряет связь, и препятствует ей в одно и то же время. Отрицательное воздействие идет от прерываний со стороны других работников из-за нахождения в помещении с открытой планировкой и от недостатка уединенности.

Никакое количество индивидуальной связи не может заменить потребность в организационных формах **групповой связи.** Групповая связь требует комнат для собрания групп, должным образом оборудованных и размещенных, в легко доступных местах, и которые можно использовать как для формальных, так и для неофициальных встреч.

Связь в разрешении конфликтов

Конфликты в коллективах неизбежны, и связь является основным средством их разрешения. Известны пять технологий для **разрешения конфликтов.** Первые три производят длительный эффект; ос-

тальные две дают только временный результат. Эти технологии следующие:

– принуждение;
– компромисс;
– конфронтация;
– сглаживание;
– изъятие.

Принуждение в разрешении конфликта делает то, что оно и предполагает. Оно воздействует на членов коллектива, которые находятся в конфликте. Принуждение возможно в случае зависимости начальник-подчиненный. Принуждающая сторона должна принять решение, но другая, вероятно, не будет согласна с этим. Начальник – это менеджер проекта или руководитель проекта. Решения, которые воздействуют на людей, обычно увеличивают авторитет менеджера проекта (если не используются слишком часто), но ослабляют обаяние и влияние руководителя проекта. Передача такого решения заинтересованным сторонам может быть сделана в форме связи, которая не будет из категории «один на один».

В противоположность этому **компромисс** требует связь категории «один на один». Он требует, чтобы стороны конфликта высказали свое мнение и достигли средневзвешенного решения без победителей или проигравших. Поскольку компромисс предполагает согласование, решение может быть длительным.

Конфронтация – наиболее эффективное и наиболее желаемое решение, когда существует один правильный ответ на конфликт и когда конфликт может быть связан с определенной проблемой (например, столкновение индивидуальностей). Конфронтация называется также решением проблемы. Как только факты установлены и решение проблемы определено, это решение может быть принято, и конфликт исчезает.

Сглаживание – в какой-то мере временный путь решения конфликта, в котором проблема, подкрепляющая конфликт, выглядит менее важной, чем есть на самом деле. По этой причине масштаб конфликта уменьшается. Сглаживание часто является примером «промывания мозгов». Вероятно, что конфликт позже снова возникнет, причем с увеличенной силой. Только в некоторых удачных случаях, когда проблема устраняется другими, казалось бы, несвязанными решениями, сглаживание обеспечивает постоянное решение конфликта.

Изъятие – другой временный способ разрешения конфликта, в котором одна сторона не видит никакого смысла в обсуждении конфликта когда-либо в дальнейшем. Это, несомненно, самый плохой результат разрешения конфликта, по крайней мере, пока обе стороны остаются в коллективе: мало того, что связь будет не в состоянии разрешить проблему, но, кроме того, изъятие приведет к ослаблению связи между членами коллектива также и на всех других проектных проблемах.

Создание коллектива

Привлечение и мотивация людей (раздел 5.1.1), а также связи проекта (раздел 5.1.2) – две доминирующие проблемы в **создании коллектива** (называемом также **построением коллектива).** Другие проблемы включают организацию коллектива, размещение, обучение, аттестацию, финансовые отчеты, внешнюю обратную связь и т. д.

Коллективы проходят четыре стадии создания:

1) формирование;
2) волнение;
3) нормализация;
4) выполнение.

Формирование коллектива начинается с первой встречи служащих на проекте. Это должна быть «строго деловая» встреча, на которой представляются члены коллектива, объясняются их роли, представляется график проектных работ, поднимаются начальные проблемы, описываются методы и инструментальные средства и т. д. В более позднее время может последовать ряд неофициальных встреч, чтобы «сломать лед».

Неизбежно, если имеется поле сложных проблем, коллектив входит в стадию **волнения.** Некоторые проблемы проекта кончаются конфронтацией и конфликтами. Источники проблем могут быть разнообразные – планы, приоритеты, управление, деньги, методика, конкретные лица и т. д. На этой стадии определяются лидеры и аутсайдеры.

Следующая стадия – **нормализация.** Подталкивание и выталкивание уступают место сотрудничеству. Члены коллектива знают свои места в проекте. Они занимаются проблемами, а не отношением друг с другом. Совместные решения достигаются без трений. Зарождается дружба.

Стадия **выполнения** достигается только в больших коллективах. На этой стадии эффективность и результативность коллектива

лучше, чем на стадии нормализации. В коллективе имеется большое понимание и очень сильные побуждающие мотивы для успеха. Все члены коллектива имеют ощущение принадлежности к общему делу и стоящей цели, так же, как и высокое удовлетворение работой.

Создание коллектива включает его **организацию.** Имеются многочисленные модели, предложенные для организации коллектива, но практически редко можно увидеть коллективы, построенные в точном согласии с теоретическими моделями. Это, вероятно, отражает уникальность проектов ПО, различия в жизненных циклах ПО, специфику организационных культур и другие подобные проблемы.

Создание коллектива – длительный процесс. Оно требует постоянного подкрепления. Даже небольшие изменения в структуре коллектива, принципах работы, проектных направлениях могут вынудить начать заново цикл из четырех стадий. Большой коллектив может превратиться в неработоспособную группу служащих. Чтобы противодействовать этому, создание коллектива должно рассматриваться как важная и составная часть совершенствования процесса создания ПО. При создании коллектива необходимо определить операции совершенствования работы, включая рабочую среду, размещение служащих в рабочих помещениях, планы обучения и т. д. Существенный аспект создания коллектива – оценка работы. Хорошая работа должна быть публично признана после того, как это произошло, но без установления премий и наград, которые, вероятно, вызовут персональные напряженные отношения в коллективе.

5.6.4. Управление рисками

Управление рисками – важный момент в планировании и отслеживании ПО. Проект может быть намного далее продвинут в смысле плана и бюджета, нежели в смысле проблем риска, потому что большинство их все еще требует решения. Скажем, например, что проект может быть на 80 процентов готов, но риск его неудачи может все еще составлять 50 процентов. Даже хуже, проект может быть закончен и представлен пользователям, в то время как некоторые проблемы все еще останутся нерешенными. Например, может быть весьма вероятно, что работа системы недопустимо ухудшится при случайных пиковых рабочих нагрузках или с ростом БД.

Стратегии управления рисками находятся в диапазоне от **реактивных** (reactive) до **упреждающих** (proactive). Реактивные стратегии рисков задерживают их обработку, пока те не станут в проекте крити-

ческими факторами. Упреждающие стратегии риска предлагают воздействовать на риски, как только они возникают или даже в ожидании их возникновения. Между этими двумя подходами имеются стратегии, которые поощряют упреждение, адресованное только наиболее неотложным проблемам риска в смысле потенциальных потерь (нежелательных воздействий) и в смысле срочности решения.

Хотя упреждающие подходы и одобряются, возможны нежелательные последствия «слишком быстрого погружения» в проблему:

Не каждый симптом заканчивается проблемой (следовательно, ресурсы могут быть израсходованы на проблему, которая никогда бы не возникла).

Функциональные руководители могут относиться к проблеме несерьезно и могут выработать неправильное мнение относительно сообщения менеджеров/ руководителей проекта о проблеме как о ложной тревоге и паникерстве.

Слишком быстрая реакция или принятие действий без полной информации может ухудшить проблему.

Время и терпение могли бы дать лучшее решение.

В этом разделе рассматривается процесс управления рисками. Типичный процесс включает, по крайней мере, три стадии:

– идентификация рисков;

– оценка рисков;

– обработка рисков.

Идентификация рисков

Идентификация рисков – утомительный процесс составления списка возможных проблем риска, полученных методом мозгового штурма и с использованием совета экспертов и опыта менеджеров. Список должен быть преднамеренно ограничен в размере, например, до 30 рисков. Риски с низким потенциальным вредом и с низкой вероятностью возникновения могут быть исключены из списка.

Чтобы помочь с идентификацией, сначала должны быть определены типы рисков. Обычно это следующие типы рисков:

– продукт проектирования (то есть риски из-за размера проекта, уникальности, сложности и т. д.);

– процесс проектирования (из-за неправильного учета или неточного следования процессу);

– люди (из-за столкновений индивидуальностей, некачественного управления, распределения людских ресурсов и т. д.);

– бизнес (из-за изменений в бизнес-условиях, политике и т. д.);

– организационный (из-за реструктурирования, финансовых трудностей и т. д.);

– технология (из-за несоответствий технологий или изменений технологий);

– инструментальные средства (из-за неспособности инструментальных средств разрабатывать нужное ПО, преобразования фирм-продавцов инструментальных средств и т. д.);

– клиенты: (из-за трудностей связи, недостатка интереса к проекту и т. д.);

– требования (из-за изменений, незавершенности, неточно указанных требований и т. д.);

– другие проекты (из-за зависимости от воздействия других проектов);

– внешние факторы (из-за государственного регулирования, юридических воздействий и т. д.).

Конкретные риски сильно меняются от проекта к проекту. Типы рисков помогают в их обнаружении. Практически организации используют различные контрольные списки рисков, просматривают их и точно определяют те риски, которые характерны для проекта. Сомервиль предлагает краткий список частых рисков:

– реорганизация штата (потеря опытных работников во время работы над проектом);

– изменение управления (новое управление устанавливает другие приоритеты);

– непригодность аппаратных средств (аппаратные средства не доступны, когда это необходимо);

– изменение требований (большее изменение требований, чем предусмотрено в плане);

– задержки спецификаций (задержки со спецификациями требуемых компонентов и интерфейсов);

– недооценка размера (система оказывается больше и сложнее, чем ожидалось);

– недостатки CASE-средств (неспособность CASE-средств в достаточной мере помочь разработчикам);

– изменение технологии (изменения в выбранной технологии проекта);

– конкуренция изделия (конкурирующие изделия угрожают жизнеспособности проекта).

Оценка рисков

Оценка рисков (называемая также анализом рисков, расчетом рисков или проектированием рисков) анализирует идентифицированные риски двумя способами:

– возможностью или вероятностью возникновения риска;

– отрицательным воздействием на проект.

Возможность может быть задана одним из пяти значений: очень вероятно, весьма возможно, вероятно, маловероятно, вряд ли возможно. С другой стороны, **вероятность** может принимать одно из девяти значений, то есть, в десятках процентов: 10 процентов, 20 процентов и т. д. до 90 процентов. Риск со 100%-й вероятностью – это уже не риск, а уверенность, т. е. проектное ограничение или требование.

Отрицательное **воздействие** риска может использовать масштаб с пятью значениями типа катастрофического, критического, серьезного, терпимого, минимального. Решение должно быть принято в зависимости от стандартных результатов рисков, в отношении которых может быть оценено отрицательное воздействие. Обычное множество таких стандартных результатов или **компонентов риска** следующее:

– риск плана (что не удастся твердо придерживаться плана);

– риск бюджета (что произойдет перерасход бюджета);

– риск использования (что у изделия не будет спроса для использования);

– риск возможности сопровождения (что изделие не будет ремонтопригодно или расширяемо).

Чтобы иметь возможность оценить риски, отрицательное воздействие должно быть нормализовано и выражено в денежных величинах. Тогда можно вычислить **подверженность риску** для каждого риска, умножая возможность/ вероятность на воздействие.

Подверженность риску можно также получить, используя графическое расположение, как показано на рис. 5.6, где одна из осей указывает масштаб возможности/ вероятности риска, а другая – масштаб воздействия риска. Риски показываются в виде точек на пересечениях между величинами возможности/ вероятности и воздействия. Риски под кривой выделения игнорируются в дальнейшем управлении рисками. Рисками выше кривой выделения необходимо дальше управлять. Чтобы изменить число рисков, которыми следует управлять, можно переместить кривую выделения в направлении стрелок.

Рис. 5.6 предполагает, что воздействие риска и возможность/ вероятность риска взвешены одинаково в оценке важности рисков. Из тринадцати рисков, изображенных на рис. 5.6, семь находятся выше

кривой выделения и ими будут в дальнейшем управлять. Шесть рисков находятся под кривой и в дальнейшем будут игнорироваться управлением рисками. Перемещение кривой выделения в направлении стрелок, показанных на рис. 5.6, изменяет число рисков, которые нужно рассмотреть для дальнейшего управления.

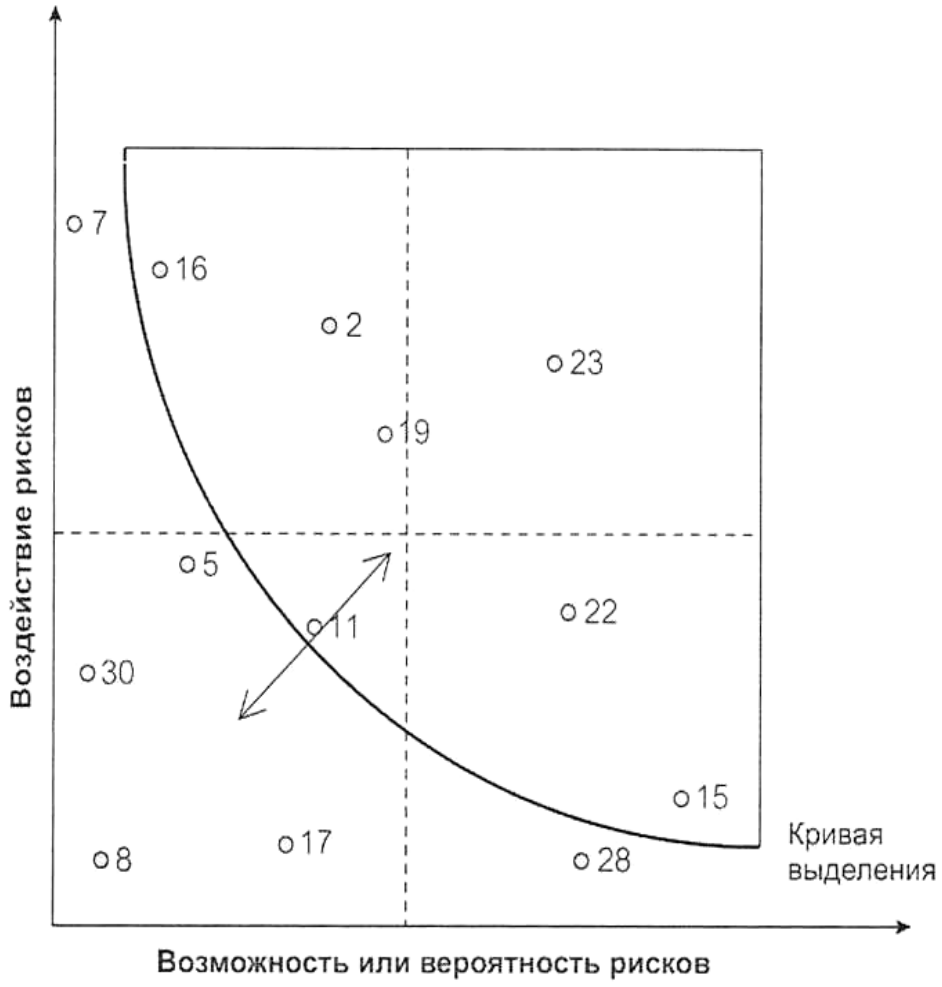


Рис. 5.6. График оценки опасности рисков

Как было упомянуто, риски изменяются во времени, и их следует постоянно отслеживать. Результаты отслеживания риска используются для корректировки планов рисков, после чего цикл планирования-отслеживания продолжается. Планирование рисков и стратегия отслеживания могут формировать отдельный **план управления рисками** или могут быть включены в общий план управления проектом.

Кроме документа, который определяет операции, связанные с управлением рисками, план управления рисками вряд ли будет

бумажным документом. Это скорее **БД рисков** – средство/ приложение БД, позволяющее по своему желанию размещать, искать, задавать приоритеты, сортировать, корректировать данные и формировать отчеты. БД рисков используется для всех трех стадий управления рисками: идентификация, оценка и обработка.

Предполагая упреждающий подход к управлению рисками, **обработка рисков** включает комбинацию трех стратегий:

– предотвращение рисков:

– минимизация рисков:

– планирование непредвиденных обстоятельств.

Стратегия **предотвращения рисков** пытается устранить возможность (вероятность) появления риска. Например, если имеется риск потери ключевого члена коллектива из-за его/ ее недовольства относительно гибкости рабочего дня, можно предложить такому человеку работать в некоторые дни дома.

Стратегия **минимизации риска** пытается уменьшить воздействие риска, когда тот появляется. По отношению к предыдущему примеру, чтобы минимизировать воздействие потери штатного работника, может быть обучен другой член коллектива, чтобы соответствовать знаниям и навыкам члена коллектива, который может оставить работу.

Стратегия **планирования непредвиденных обстоятельств** предполагает, что риск может стать действительностью и необходимо знать, как реагировать, когда это случится. План на случай возникновения непредвиденных обстоятельств определяет последовательность действий, которые будут предприняты, когда другие стратегии не смогли ничего сделать, и риск стал фактом. Например, ключевой член коллектива уезжает с уведомлением за две недели, и нет никакой немедленной доступной замены. План на случай возникновения непредвиденных обстоятельств может тогда определить, что служащий (человек, находящийся на контракте) должен быть нанят, бюджет и план должны быть соответственно отрегулированы, а отбывающий служащий должен потратить оставшиеся две недели, чтобы задокументировать состояние своей работы.

5.6.5. Управление качеством

Управление качеством ПО – всеобъемлющая операция, которая переплетается с большинством других операций управления и охватывает весь жизненный цикл разработки и ее процесс. Оно переплетается и охватывает, однако, все же это независимая операция.

Коллектив управления качеством должен быть отделен коллектива разработчиков и иметь свои собственные каналы информации. Управление качеством должно быть независимым от управления проектом. План управления качеством должен иметь свой собственный бюджет и график, по крайней мере, настолько, насколько интересует сама гарантия качества.

Управление качеством было предметом интенсивных усилий по стандартизации. Модель зрелости возможностей (Capability Maturity Model – CMM), рассматриваемая в начале этой главы, является дефакто стандартом для процесса управления качеством. Международная организация по стандартизации (International Organization for Standardization) приняла серию стандартов качества ISO 9000. Стандарты применимы к любым отраслям промышленности и всем типам областей операций, включая разработку ПО. Вспомогательный документ, известный как ISO 9000-3, поясняет использование ISO 9000 для разработки ПО.

Совсем недавно ключевые стандарты в пределах серии стандартов ISO 9000 были слиты в единый стандарт, известный как ISO 9001:2000. Акцент стандарта направлен на управление ключевыми процессами с целью непрерывного их улучшения (что аналогично уровню 5 CMM). Как и в случае CMM-сертификации, ISO обеспечивает возможность для организаций зарегистрировать свое соответствие процессам, определяемым ISO.

Стандарты управления качеством в первую очередь связаны с определением процессов обеспечения качества, которые гарантировали бы надлежащее качество изделий. Эти процессы отличаются от механизмов качества, связанных с управлением качеством в изделиях. Соответственно, *гарантия качества* (раздел 5.3.3) отличается от *контроля качества* (раздел 5.3.2).

Показатели качества программного обеспечения

Различные пользователи (и различные разработчики) имеют разные представления относительно того, что составляет **показатели качество ПО.** Имеется много желаемых показателей качества, и в зависимости от проекта ПО некоторые из них могут быть более важными, чем другие. Показатели качества ПО должны быть идентифицированы, определены и классифицированы.

Основная классификация делит показатели качества ПО на **показатели качества изделий** и **показатели качества процессов.** Обычно нельзя произвести качественное изделие без качественного

процесса, который определяет производственные операции. Цель процесса – изделие. Следовательно, хотя показатели качества могут быть приписаны процессу как таковому, все они могут быть непосредственно связаны с качеством изделия. Обратное также верно.

С точки зрения управления качеством качество изделия должно быть гарантировано и управляться не только для **конечного изделия,** поставляемого клиенту, но также и (и прежде всего) для всех промежуточных **рабочих изделий** (которые известны все вместе под термином **продукты).** Управление качеством в рабочих изделиях требует поддержки инструментальных средств управления конфигурацией, способных хранить и управлять многочисленными продуктами, их версиями и отношениями между ними.

Специалисты предлагают превосходный набор типичных показателей качества программных продуктов и процессов. Эти показатели следующие:

– корректность;
– надежность;
– ошибкоустойчивость (робастность);
– выполнение функций;
– применимость;
– понятность;
– удобство сопровождения (ремонтопригодность);
– масштабируемость (способность к развитию);
– возможность многократного использования;
– мобильность;
– способность к взаимодействию;
– производительность;
– своевременность;
– видимость.

Корректность (correctness) – ряд формальных свойств ПО, которые устанавливают непосредственное соответствие между программным продуктом и его функциональными спецификациями. Существуют две проблемы с показателем корректности. Во-первых, имеются мощные аргументы в пользу известного высказывания, что невозможно доказать корректность программы, потому что невозможно гарантировать отсутствие ошибок в таких сложных изделиях, как ПО. Во-вторых, функциональные спецификации редко определяются с достаточной точностью и стабильностью, чтобы служить исчерпывающим ориентиром в определении показателя правильности. Поэтому было бы лучше иметь правильное ПО при неправильных требованиях.

Надежность (reliability), также называемая **функциональной надежностью** (dependability), – свойство ПО, обеспечивающее к нему доверие пользователей, потому что оно ведет себя правильно и таким образом, как ожидают пользователи. В идеале понятие надежности ПО должно включать понятие корректности. В то время как это истинно в традиционных технических дисциплинах, в разработке ПО надежные программы могут содержать «известные ошибки». Даже если приходится столкнуться с новыми ошибками, ПО можно все еще считать (достаточно) надежным.

Ошибкоустойчивость (robustness) подразумевает, что ПО вряд ли подведет или испортится, или, по крайней мере, будет безвозвратно разрушено. Ошибкоустойчивость дополняет понятия корректности и надежности. Вместе эти три качества определяют, выполняет ли ПО свои функции так, как ожидалось. Кроме того, все три свойства могут характеризовать качество как изделия, так и процесса.

Выполнение функций (performance) означает работать удовлетворительно в соответствии с заданными целями. Обычной целью является время отклика системы ПО, которое определяет, как долго пользователь готов ждать, пока система ответит на его запрос. В отличие от многих других качеств выполнение функций – черно-белое качество: или система обладает им, или нет.

Применимость (usability) касается дружелюбия к пользователю ПО; насколько легко пользователю работать с ним. Это – очень субъективное свойство. Что является дружественным и легким для одного человека, может быть недружелюбным и сложным для другого. Качество применимости – основная характеристика при проектировании пользовательского интерфейса для системы. Как и в традиционных технических дисциплинах, чем более стандартный и типовой пользовательский интерфейс, тем он более пригоден. Новаторские проекты пользовательских интерфейсов могут выглядеть прелестно, но быть ужасными в употреблении, особенно для новичка, однако грамотного в компьютерном отношении пользователя. **Понятность** (understandability) означает легкость, с которой могут быть проанализированы и поняты внутренняя структура и поведение ПО эксплуатационным персоналом. Понятность является условием для удобства сопровождения.

Удобство сопровождения (maintainability) ПО отличается от удобства сопровождения других технических изделий. Части ПО не ломаются из-за длительного использования, скачков напряжения и других подобных причин. Удобство сопровождения при разработке

ПО является всеобъемлющей концепцией, означающей решение каждой из следующих трех задач: 1) исправление ошибок и недостатков ПО, 2) приспособление ПО к новым требованиям и 3) совершенствование ПО для получения новых качеств. В узком смысле слова, принятом здесь, удобство сопровождения ограничено только первым пунктом. Это называется **ремонтопригодностью** (repairability).

Масштабируемость (scalability) или **способность к развитию** (evolvability) означает легкость, с которой ПО может быть увеличено или развито в ответ на повышение требований к его функциональным возможностям. Масштабируемость может быть достигнута лишь в системах с ясным, понятным структурным проектированием. Как только структура системы ухудшается в результате сопровождения, масштабируемость уменьшается. Понятность, удобство сопровождения и масштабируемость известны под объединенным названием **возможности сопровождения** (supportability).

Возможность многократного использования (reusability) ПО определяет уровень, на котором компоненты ПО могут использоваться для конструирования других изделий. ПО может многократно использоваться двумя способами. Оно может использоваться как **компонент** другого изделия или может использоваться как **шаблон,** который следует настроить, чтобы создать другое изделие. Возможность многократного использования применяется также и к процессам создания ПО.

Мобильность (portability) означает, что ПО может выполняться на различных платформах аппаратных средств (программного обеспечения) без какой-либо модификации или после выполнения незначительной настройки или параметризации. Мобильность имеет экономическое значение. Это качество существенно для системного ПО и в меньшей степени для прикладного ПО.

Способность к взаимодействию (interoperability) определяет способность ПО сосуществовать или работать вместе с другим ПО, возможно, даже с будущим ПО, которое еще не существует. Подобно мобильности способность к взаимодействию существенна для системного ПО. Это качество обеспечивает такое ПО, которое известно, как **открытые системы.**

Производительность (productivity) – показатель качества процесса. Производительность определяет скорость, с которой процесс позволяет создать ПО, имея некоторое количество ресурсов. Производительность – мера эффективности и выполнения функций процесса.

Своевременность (timeliness) – другой показатель качества процесса, который определяет способность процесса создать ПО вовремя. «Вовремя» обычно означает согласно плану исходных данных (пересмотренный план, согласно которому изделие было поставлено, вряд ли будет своевременным). В случае коммерческих изделий своевременность может означать «вовремя для рынка».

Наконец, **видимость** (visibility) – также показатель качества процесса. Видимый процесс – прозрачный процесс – процесс с ясно определенными и задокументированными стадиями и операциями. Видимый процесс является условием хорошего проекта и управления рисками. Он является также условием для организации, чтобы получить CMM-свидетельство или регистрацию ISO.

Контроль качества

Контроль качества главным образом обеспечивается тестированием качества изделия, в противоположность **гарантии качества**, которая характеризуется непосредственным обеспечением качества изделия. Контроль качества имеет реактивный характер. Это в большой степени – упреждающее действие. Контроль качества представляет собой в основном оперативные и тактические действия. Гарантия качества имеет сугубо стратегический аспект. Иногда контроль качества рассматривается как часть гарантии качества.

Иногда контроль качества приравнивают к управлению вариациями. Управление вариациями – концепция, используемая в производстве, где производятся многократные копии одного и того же изделия, и они должны быть идентичны. Копия, которая имеет вариации по сравнению с моделью (шаблоном) копии, не удовлетворяет проверке качества.

В разработке ПО многократные копии программного продукта обычно не бывают. Большинство прикладных программных продуктов существует в единственной копии. Когда все же производятся многократные копии, типа коммерческих систем ПО, дублирование является простым вопросом создания компакт-дисков или других носителей данных. Часто коммерческое распределение ПО даже не требует производства многократных копий – ПО можно просто скачать с Web-сайта продавца ПО.

Тестирование ПО

Имеется параллель между контролем качества ПО и управлением вариациями. Контроль качества – нечто вроде обнаружения вариа-

ций в работе или конечном изделии по сравнению с данной спецификацией. Спецификация может включать выполнение функциональных требований и/ или удовлетворение показателей качества, имеющихся в списке раздела 5.3.1. Цель контроля качества ПО состоит в том, чтобы подвергать ПО частым испытаниям для устранения проблем как можно раньше. Во многих случаях контроль качества ПО синонимичен тестированию программного обеспечения.

Как было сказано выше, тестирование – это не отдельная стадия жизненного цикла разработки. Подобно управлению качеством, тестирование – всеобъемлющая операция, которая применяется ко всем стадиям жизненного цикла. Естественно, тестирование наиболее интенсивно, когда доступен код. Однако, как это ни парадоксально, тестирование кода – не обязательно наиболее ценный вид тестирования в терминах плана и бюджета проектных работ.

Как говорилось в старых пособиях IBM, относительная стоимость исправления ошибки растет почти по экспоненте в зависимости от стадии жизненного цикла, в которой эта ошибка была обнаружена. Исследование IBM установило факт, что стоимость исправления ошибки может изменяться в десятки раз на разных стадиях жизненного цикла (см. рис. 5.6). Следовательно, принимая, что исправление ошибки, найденной на стадии анализа требований, стоит 1 доллар, исправление той же самой ошибки стоило бы целых 10 долларов на стадии проектирования системы, 100 долларов, если она обнаружена на стадии реализации, и целых 1000 долларов, если она обнаружена после того, как система будет развернута пользователями.

Эта повсеместно принятая истина об относительной стоимости исправления ошибок дает дополнительный стимул и определяет важность контроля качества и тестирования операций. Тестирование имеет различные формы в зависимости от стадии разработки. На стадии анализа требований тестирование старается установить полноту требований, чтобы устранить несовместимости и противоречия, согласовать с пользователями все необходимые опытные образцы ПО и т. д. Во время проектирования системы тестирование всесторонне проверяет различные модели проекта, анализирует по спецификациям продукты проекта, гарантирует, что все требования учтены, и т. д. На стадии реализации и позже ведущее положение занимает тестирование кода.

Имеется интересная зависимость между продуктами разработки ПО и тестированием кода. Зависимость состоит в том, что более поздние стадии тестирования концентрируются на проверке более

ранних продуктов разработки. Например, тестирование интеграции проводится со спецификациями структурного и детального проекта.

Однако приемочные испытания подчеркивают текущие требования клиента, которые, будем надеяться, не изменились по сравнению с тем, как они были задокументированы для целей проектирования и анализа. Если они все же изменились, приемочные испытания могут дать отрицательную обратную связь от клиента, и возникшие проблемы должны быть решены между клиентами и менеджерами проекта.

Технологии тестирования

Имеется много различных **технологий тестирования,** которые можно комбинировать, чтобы получить лучший возможный охват и результаты тестирования. Независимо от того, насколько обширно тестирование, оно не может быть исчерпывающим и гарантировать корректность программы или системы. Невозможно проверить все возможные входные данные или все пути выполнения кода.

Технологии (стратегии) тестирования могут классифицироваться по разным критериям. Дальнейшее обсуждение основано на пяти критериях: I) видимость, 2) автоматизация, 3) разделение, 4) охват и 5) использование сценариев. Критерии не являются независимыми. Например, две технологии разделения применяются главным образом к тестированию «черного ящика», но не применимы к тестированию «белого ящика». Технологии использования сценариев (регрессионные тесты и испытательные тесты) являются автоматическими испытаниями.

Тестирование «черного ящика» (или **тестирование на основе технических требований)** предполагает, что испытатель при тестировании не знает или игнорирует внутреннюю работу программной единицы. Испытуемая единица рассматривается как «черный ящик», который имеет некоторый вход и формирует некоторый результат. Тестирование выполняют, подавая входные данные на испытуемую единицу и проверяя, что получен ожидаемый результат. Поскольку никаких знаний устройства не требуется, тестирование «черного ящика» может выполняться пользователями. Соответственно, оно является главной технологией приемочных испытаний.

Технология «черного ящика» проверяет функциональные возможности системы или функциональные выходы системы, если быть точными. Она также применима для тестирования ограничений, типа выполнения функций или безопасности системы. Особая цель тести-

рования «черного ящика» – проверка **нарушений функциональных возможностей**, т. е., когда ожидаемые функциональные требования не были реализованы в ПО. В этом случае испытательная единица получает данные, вызывающие такие функциональные возможности, и выдает ошибку, потому что нет необходимого кода, выполняющего требуемые действия.

Тестирование «белого ящика» (или **тестирование на основе кода)** – противоположность тестированию «черного ящика». Испытуемая единица рассматривается как белый ящик или, точнее, прозрачный **стеклянный ящик,** который показывает свое содержание. В этом подходе испытатель изучает код и выбирает входные данные, которые могут «реализовать» выбранные пути выполнения кода. Тестирование «черного ящика» должно обычно сопровождаться тестированием «белого ящика», чтобы точно определить ошибки, обнаруженные при тестировании на основе технических требований.

В отличие от тестирования «черного ящика», тестирование «белого ящика» не ограничивается тестированием кода. Оно одинаково подходит и для других продуктов разработки, типа моделей проекта и документов спецификации. Тестирование этого вида использует технологии просмотра, типа сквозного контроля и инспекции. Интересным результатом, который можно получить с помощью «белого ящика», но не с «черным ящиком», является выявление существования **«мертвого кода».** Мертвый код – это операторы программы, которые нельзя выполнить, то есть, они никогда не будут использоваться независимо от того, каким будет вход.

Технологии разделения эквивалентностей и граничных условий применяются главным образом к тестированию «черного ящика», хотя и при тестировании «белого ящика» также можно получить положительный результат. Это определенные подходы к технологии «черного ящика», чтобы противодействовать невозможности проведения исчерпывающих испытаний.

Контрольные вопросы и задания

1. Объясните, как программная инженерия и системная инженерия соотносятся друг с другом. Является ли это отношением включения или пересечения? Может быть, эти две концепции вообще не имеют друг к другу никакого отношения?

2. Каковы пять главных аспектов программной инженерии? Можете ли вы сказать, что любой случай программной инженерии обязательно охватывает все эти аспекты?

3. Что мы подразумеваем, когда говорим, что информационные системы являются социальными системами? Может ли быть система социальной? Рассмотрите следующее объяснение термина «социальный» – «касающийся действий, когда вы встречаетесь и проводите время с другими людьми, и которые происходят в то время, когда вы не работаете».

4. Каковы основные различия между БД и хранилищем данных? Как эти две концепции связаны с уровнями управления в организации?

5. Что мы подразумеваем под прямым и обратным проектированием? Как это касается программирования?

6. Программная инженерия связана с моделированием систем. Что моделируется в процессе программной инженерии? Как это соотносится с понятием абстракции? Имеет ли смысл говорить о моделировании программ?

7. Как вы понимаете различие между функциональным и объектно-ориентированным подходами к разработке системы?

8. Какой фактор наиболее важен в определении сложности современной объектно-ориентированной системы? Какова основная технология управления сложностью и ее уменьшением? Объясните.

9. Объясните отношение между анализом требований и техническими требованиями.

10. Каково различие между определением требований и техническим заданием? Объясните.

11. Проведите линию раздела между анализом требований и проектированием системы. Когда анализ становится проектированием?

12. Каковы главные подходы в технологии тестирования?

13. Объясните использование заглушек и драйверов в тестировании интеграции.

14. Что такое управление рисками? Какая из моделей жизненного цикла наиболее явно использует управление рисками? Объясните.

15. Что такое выполнимые спецификации? Какая модель жизненного цикла использует выполнимые спецификации как свой основной момент? Объясните.

16. Каковы наиболее необычные аспекты быстрой разработки (необычные по сравнению с другими итеративными подходами)? Объясните.

17. Объясните уровни зрелости процесса CMM. Каковы основные факторы для улучшения уровней зрелости?

18. Что такое сетевая организация? Каковы главные проблемы управления людьми в пределах сетевой организации? Свяжите ваши объяснения с процессами создания коллектива и с подходящими мотивационными теориями.

19. Обсудите взаимодействие между формами связи и линиями связи.

20. Обсудите взаимодействие между формами связи и разрешением конфликтов.

21. Насколько полезны реактивные стратегии в управлении рисками? Объясните.

22. Как подсчитывается подверженность риску? Что такое контрольная точка риска?

23. Каковы стратегии обработки рисков в упреждающем управлении рисками? Как они применяются? Приведите пример.

24. Объясните различия между гарантией качества и контролем качества. Какие главные технологии управления качеством используются в контроле качеством и в гарантии качества?

25. Объясните качество возможности сопровождения.

26. Считают, что тестирование «черного ящика» и «белого ящика» являются дополнением друг другу. Объясните.

27. Каково отношение между регрессионными тестами и испытательными тестами? Могут ли они использоваться совместно? Объясните.

28. Объясните концепции контрольного примера, сценария тестирования и тестового набора. Как эти концепции связаны?

29. Как вы понимаете культуру качества? Сравните, как операции контроля качества и гарантии качества вносят вклад в культуру качества.

Глава 6

СТАНДАРТЫ В ОБЛАСТИ ПРОГРАММНОЙ ИНЖЕНЕРИИ

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.

Слово 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).

Стандарт может быть разработан:

– на материально-технические предметы (продукцию, эталоны, образцы веществ);

– нормы, правила, требования организационно-методического и общетехнического характера.

В настоящее время стандартизация распространяется на все сферы человеческой деятельности: науку, образование, технику, промышленное и сельскохозяйственное производство, строительство, здравоохранение, транспорт и, как мы говорили раньше, системная и программная инженерия.

В данной главе мы кратко рассмотрим три вида стандартов:

1) Федеральный государственный образовательный стандарт по направлению 09.04.04 Программная инженерия;

2) Профессиональные стандарты из группы 06 Связь, информационные и коммуникационные технологии;

3) Стандарты системной и программной инженерии.

6.1. Регламентирующие документы для подготовки магистров по программной инженерии

6.1.1. Федеральный государственный образовательный стандарт

Обучение магистров по программной инженерии в РФ в настоящее время регламентируется федеральным государственным образовательным стандартом по направлению 09.04.04 Программная инженерия, утвержденным приказом Министерства образования и науки Российской Федерации от 19 сентября 2017 г. № 932, который привязан к существующим профессиональным стандартам.

Рассмотрим кратко требования к выпускникам этой специальности, определяемые отечественными стандартами.

Согласно ФГОС 09.04.04 Программная инженерия, выпускник магистратуры должен обладать следующим набором универсальных компетенций:

УК-1. Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий;

УК-2. Способен управлять проектом на всех этапах его жизненного цикла;

УК-3. Способен организовывать и руководить работой команды, вырабатывая командную стратегию для достижения поставленной цели;

УК-4. Способен применять современные коммуникативные технологии, в том числе на иностранном(ых) языке(ах), для академического и профессионального взаимодействия;

УК-5. Способен анализировать и учитывать разнообразие культур в процессе межкультурного взаимодействия;

УК-6. Способен определять и реализовывать приоритеты собственной деятельности и способы ее совершенствования на основе самооценки.

Кроме того, выпускник магистратуры должен обладать следующим набором общепрофессиональных компетенций:

ОПК-1. Способен самостоятельно приобретать, развивать и применять математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте;

ОПК-2. Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач;

ОПК-3. Способен анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями;

ОПК-4. Способен применять на практике новые научные принципы и методы исследований;

ОПК-5. Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем;

ОПК-6. Способен самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе в новых областях знаний, непосредственно не связанных со сферой деятельности;

ОПК-7. Способен применять при решении профессиональных задач методы и средства получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе, в глобальных компьютерных сетях;

ОПК-8. Способен осуществлять эффективное управление разработкой программных средств и проектов.

Что касается конкретных профессиональных компетенций, то их выбор остается за высшим учебным заведением в соответствии с направленностью подготовки магистрантов, и в данной работе они не рассматриваются.

6.1.2. Профессиональные стандарты, ассоциированные с ФГОС 09.04.04 Программная инженерия

Профессиональные стандарты разрабатываются по инициативе Агентства стратегических инициатив РФ (АСИ) в рамках проекта «Национальная система компетенций и квалификаций (НСКК)», который предусматривает разработку системы, описывающей и формирующей отношения граждан, бизнес-структур и государственных структур по созданию, развитию, накоплению, воспроизводству, оценке и защите компетенций в целях повышения конкурентоспособности страны.

Поэтому в федеральном образовательном стандарте имеются ссылки на следующие профессиональные стандарты, наиболее близкие к требованиям ФГОС.

1. Профессиональный стандарт 06.003 «Архитектор программного обеспечения», утвержденный приказом Министерства труда и социальной защиты Российской Федерации от 11 апреля 2014 г. № 228н (зарегистрирован Министерством юстиции Российской Федерации 2 июня 2014 г., регистрационный № 32534), с изменением, внесенным приказом Министерства труда и социальной защиты Российской Федерации от 12 декабря 2016 г. № 727н (зарегистрирован Министерством юстиции Российской Федерации 13 января 2017 г., регистрационный № 45230);

2. Профессиональный стандарт 06.017 «Руководитель разработки программного обеспечения», утвержденный приказом Министерства труда и социальной защиты Российской Федерации от 17 сентября 2014 г. № 645н (зарегистрирован Министерством юстиции Российской Федерации 24 ноября 2014 г., регистрационный № 34847), с изменением, внесенным приказом Министерства труда и социальной защиты Российской Федерации от 12 декабря 2016 г. № 727н (зарегистрирован Министерством юстиции Российской Федерации 13 января 2017 г., регистрационный № 45230);

3. Профессиональный стандарт 06.028 «Системный программист», утвержденный приказом Министерства труда и социальной защиты Российской Федерации от 5 октября 2015 г. № 685н (зарегистрирован Министерством юстиции Российской Федерации 20 октября 2015 г., регистрационный № 39374).

Итак, мы видим, что, требования к квалификации выпускника вуза в настоящее время определяется двумя видами документов: государственными образовательными стандартами и профессиональными стандартами.

Проблема заключается в разном идеологическом и методическом подходе к описанию требований к специалистам в указанных выше документах, которые никак между собой не согласованы, хоть имеют близкие цели – установление требований к уровню компетентности специалистов.

Если в образовательных стандартах формулируются требования к компетенциям специалиста (компетенция означает: знать, уметь, владеть), то профессиональные стандарты определяют трудовые функции, которые должен выполнять специалист в зависимости от должности, которую он занимает.

Учитывая сказанное, студенты в процессе освоения курса «Методология программной инженерии» должны изучить оба вида документов.

В данном пособии нет возможности рассмотреть эти стандарты, их изучение планируется на практических и семинарских занятиях.

6.2. Стандарты по системной и программной инженерии

Основным объектом стандартизации в области системной и программной инженерии сегодня являются процессы жизненного цикла создания систем, кроме того, стандартизируются методы оценки качества и зрелости этих процессов, а также способы описания системных артефактов, а именно – систем и процессов. При этом в помощь специалистам – разработчикам систем активно формируется так называемый профиль стандартов – полная система руководств, позволяющих эффективно использовать нормативно-техническое обеспечение деятельности по созданию систем.

6.2.1. Состояние отечественного нормативно-технического обеспечения по системной и программной инженерии

По мнению ведущих ИТ-специалистов, в плане методического и нормативно-технического обеспечения деятельности по созданию больших систем наша страна в настоящее время находится в режиме догоняющего развития по отношению к мировому инженерному сообществу [12].

Можно выделить две основные причины невысокого интереса отечественных специалистов к стандартам системной и программной инженерии:

– сложность освоения и применения современных стандартов;

– существенные отличия между отечественной и западной культурами разработки систем.

Сложность освоения стандартов состоит в том, что современные рекомендации по системной и программной инженерии предполагают: каждая отрасль или крупная корпорация разрабатывает на основе стандартов, подобных ISO/IEC 15288 и ISO/IEC 12207, собственные нормативные документы, адаптированные к корпоративным нуждам. Далее, с учетом полученных результатов, определяется совокупность методов и инструментов управления процессами ЖЦ при создании систем. Объем работ по подготовке подобной системы документов весьма велик, кроме того, высоки требования к квалификации занятого в этом процессе персонала. Создание и апробация необходимого крупной организации комплекта документов может занять до 3–5 лет

и потребовать многомиллионных затрат. К работе требуется привлекать большое количество высококвалифицированных специалистов. К тому же требуется организовать сопровождение полученной системы документов и обеспечить их регулярную актуализацию.

В настоящее время ситуация меняется к лучшему – развитию цифровой экономики, в том числе методов и технологий программной инженерии уделяется большое внимание как со стороны государственных структур. так и со стороны бизнеса.

В настоящее время Федеральным агентством по техническому регулированию и метрологии ведется разработка отечественных стандартов в области системной и программной инженерии, согласованных с международно-признанными стандартами.

Тем не менее, для сохранения уровня и конкурентоспособности отечественных разработок в области больших программных систем переход к использованию международных стандартов не имеет альтернативы. В этом смысле большие надежды возлагаются на новое поколение выпускников вузов, которые, освоив новые методологии и технологии программной инженерии, смогут создавать программные системы мирового уровня. Решение о начале подготовки в вузах страны бакалавров и магистров по направлению «Программная инженерия» и большой конкурс на это направление во многих вузах также говорят об этом.

Ниже мы рассмотрим наиболее известные отечественные стандарты, а также кратко ознакомимся с некоторыми международными стандартами по системной и программной инженерии.

6.2.2. Отечественные стандарты по автоматизированным системам

В Советском Союзе существовала развитая система государственных стандартов (ГОСТов). Они охватывали практически все области человеческой деятельности – промышленность, сельское хозяйство, геологию, торговлю и многие другие. Была создана система стандартов и в области информационной технологии. Отечественная практика разработки стандартов создания систем начала формироваться в конце 60-х – начале 70-х годов прошлого века и нашла отражение в комплексе стандартов ГОСТ 24 «Единая система стандартов автоматизированных систем управления», получившем дальнейшее развитие в весьма успешном и популярном комплексе стандартов на автоматизированные системы ГОСТ 34, который был разработан

в СССР в конце 80-х годов и до сих пор, несмотря на известную устарелость, применяется в нашей стране при создании систем, основанных на использовании ИТ.

Единая система стандартов автоматизированных систем управления ГОСТ 24 разрабатывалась с конца 1970-х до середины 1980-х годов и содержала спецификации, в которых устанавливалось содержание и требования к документированию результатов работ по созданию (развитию) автоматизированных систем управления (АСУ). Система стандартов ГОСТ 24 включала более 20 спецификаций, в частности, в стандартах ГОСТ 24 описывались типовые стадии ЖЦ АСУ, типовые проектные решения, способы оценки важнейших характеристик АСУ и другие положения.

В целях распространения положений ГОСТ 24 на более широкий спектр систем, в конце 1980-х годов был разработан единый комплекс стандартов на автоматизированные системы (ЕКС АС) ГОСТ 34. ЕКС АС представляет собой совокупность взаимоувязанных межотраслевых документов, обеспечивающих стандартизацию и унификацию АС всех видов, процессов их создания, функционирования и развития. Комплекс распространяется на АС и их составные части, а также на ЖЦ и документирование АС.

Следует заметить также, что требования ГОСТ 34 легли в основу требований к выпускным квалификационным работам специалистов по ИТ-технологиям, принятым во многих вузах РФ.

Структура советского стандарта

Каждый стандарт определялся следующей классификационной схемой

ГОСТ ХХ. Х ХХ – ХХ

В этой схеме:

слово ГОСТ определяет категорию стандарта;

первые две цифры определяют класс стандарта;

следующая цифра определяет классификационную группу стандартов;

следующие две цифры определяют порядковый номер стандарта в группе;

последние две цифры означают год регистрации стандарта.

Для нас наибольший интерес представляют следующие классы стандартов:

ГОСТ 19. Х ХХ – ХХ Стандарт единой системы программной документации (ЕСПД);

ГОСТ 24. Х ХХ – ХХ Единая система стандартов автоматизированных систем управления;

ГОСТ 34. Х ХХ – ХХ Информационная технология. Комплекс стандартов на автоматизированные системы.

Кроме государственных стандартов существуют еще категории руководящих документов (методических указаний), шифр РД и рекомендаций, шифр Р.

В качестве примера приведем фрагменты из ГОСТ 34-го класса. Сведения, содержащиеся в этих материалах, потребуются студентам при выполнении курсовых работ и проектов, а также выпускных квалификационных работ бакалавров и магистров.

Семейство ГОСТ 34, как уже сказано, разрабатывалось в СССР самостоятельно, как стандарт на содержание и оформление документов на программные системы в рамках Единой системы программной документации (ЕСПД) и Единой системы конструкторской документации (ЕСКД).

В последние годы акцент делается на стандарты ГОСТ, соответствующие международным стандартам. Несмотря на это, 34-я серия является важным дополнительным источником информации для разработки и стандартизации внутрикорпоративных документов и формирования целостного понимания и видения концепций жизненного цикла в области программного обеспечения.

Мы рассмотрим кратко содержание двух стандартов 34 группы, которые могут оказаться полезными в учебном процессе. Полный текст стандартов можно найти в Интернете.

6.2.3. ГОСТ 34.601–90 Автоматизированные системы. Стадии создания.

Стандарт устанавливает стадии и этапы создания автоматизированных систем (АС).

1. Общие положения

1.1. Процесс создания АС представляет собой совокупность упорядоченных во времени, взаимосвязанных, объединенных в стадии и этапы работ, выполнение которых необходимо и достаточно для создания АС, соответствующей заданным требованиям.

1.2. Стадии и этапы создания АС выделяются как части процесса создания по соображениям рационального планирования и организации работ, заканчивающихся заданным результатом.

1.3. Работы по развитию АС осуществляют по стадиям и этапам, применяемым для создания АС.

1.4. Состав и правила выполнения работ на установленных настоящим стандартом стадиях и этапах определяют в соответствующей документации организаций, участвующих в создании конкретных видов АС.

2. Стадии и этапы создания АС

2.1. В стандарте в общем случае предусмотрено 8 стадий и соответствующих этапов создания АС.

1. Формирование требований к АС.

1.1. Обследование объекта и обоснование необходимости создания АС.

1.2. Формирование требований пользователя к АС.

1.3. Оформление отчета о выполненной работе и заявки на разработку АС (тактико-технического задания).

2. Разработка концепции АС.

2.1. Изучение объекта.

2.2. Проведение необходимых научно-исследовательских работ.

2.3 Разработка вариантов концепции АС и выбор варианта концепции АС, удовлетворявшего требованиям пользователя.

2.4. Оформление отчета о выполненной работе.

3. Техническое задание.

3.1. Разработка и утверждение технического задания на создание АС.

4. Эскизный проект.

4.1. Разработка предварительных проектных решений по системе и ее частям.

4.2. Разработка документации на АС и ее части.

5. Технический проект.

5.1. Разработка проектных решений по системе и ее частям.

5.2. Разработка документации на АС и ее части.

5.3. Разработка и оформление документации на поставку изделий для комплектования АС и (или) технических требований (технических заданий) на их разработку.

5.4. Разработка заданий на проектирование в смежных частях проекта объекта автоматизации.

6. Рабочая документация.

6.1. Разработка рабочей документации на систему и ее части.

6.2 Разработка или адаптация программ.

7. Ввод в действие.

7.1. Подготовка объекта автоматизации к вводу АС в действие.

7.2. Подготовка персонала.

7.3. Комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями).

7.4 Строительно-монтажные работы.

7.5. Пусконаладочные работы.

7.6. Проведение предварительных испытаний.

7.7. Проведение опытной эксплуатации.

7.8. Проведение приемочных испытаний.

8. Сопровождение АС.

8.1. Выполнение работ в соответствии с гарантийными обязательствами.

8.2. Послегарантийное обслуживание.

2.2. Стадии и этапы, выполняемые организациями – участниками работ по созданию АС, устанавливаются в договорах и техническом задании на основе настоящего стандарта.

Допускается исключать стадию «Эскизный проект» и отдельные этапы работ на всех стадиях, объединять стадии «Технический проект» и «Рабочая документация» в одну стадию «Технорабочий проект». В зависимости от специфики создаваемых АС и условий их создания допускается выполнять отдельные этапы работ до завершения предшествующих стадий, параллельное во времени выполнение этапов работ, включение новых этапов работ.

6.2.4. ГОСТ 34.602–89 Техническое задание на создание автоматизированной системы

Стандарт устанавливает структуру и содержание ТЗ на создание автоматизированной системы.

1. Общие положения

1.1. Техническое задание на автоматизированную систему (ТЗ на АС) является основным документом, определяющим требования и порядок создания (развития или модернизации – далее создания) автоматизированной системы, в соответствии с которым проводится разработка АС и ее приемка при вводе в действие.

1.2. ТЗ на АС разрабатывают на систему в целом, предназначенную для работы самостоятельно или в составе другой системы.

Дополнительно могут быть разработаны ТЗ на части АС: на подсистемы АС, комплексы задач АС и т. п. в соответствии с требованиями настоящего стандарта; на комплектующие средства техниче-

ского обеспечения и программно-технические комплексы в соответствии со стандартами ЕСКД и СРПП; на программные средства в соответствии со стандартами ЕСПД; на информационные изделия в соответствии с ГОСТ 19.201 и НТД, действующей в ведомстве заказчика АС.

1.3. Требования к АС в объеме, установленном настоящим стандартом, могут быть включены в задание на проектирование вновь создаваемого объекта автоматизации. В этом случае ТЗ на АС не разрабатывают.

1.4. Включаемые в ТЗ на АС требования должны соответствовать современному уровню развития науки и техники и не уступать аналогичным требованиям, предъявляемым к лучшим современным отечественным и зарубежным аналогам. Задаваемые в ТЗ на АС требования не должны ограничивать разработчика системы в поиске и реализации наиболее эффективных технических, технико-экономических и других решений.

1.5. ТЗ на АС разрабатывают на основании исходных данных, в том числе, содержащихся в итоговой документации стадии «Исследование и обоснование создания АС», установленной ГОСТ 24.601.

1.6. В ТЗ на АС включают только те требования, которые дополняют требования к системам данного вида (АСУ, САПР, АСНИ и т. д.), содержащиеся в действующих НТД, и определяются спецификой конкретного объекта, для которого создается система.

1.7. Изменения к ТЗ на АС оформляют дополнением или подписанным заказчиком и разработчиком протоколом. Дополнение или указанный протокол являются неотъемлемой частью ТЗ на АС. На титульном листе ТЗ на АС должна быть запись «Действует с ...».

2. Состав и содержание

2.1. ТЗ на АС содержит следующие разделы, которые могут быть разделены на подразделы:

1) общие сведения;

2) назначение и цели создания (развития) 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 составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие;

8) требования к документированию;

9) источники разработки.

В ТЗ на АС могут включаться приложения.

2.2. В зависимости от вида, назначения, специфических особенностей объекта автоматизации и условий функционирования системы допускается оформлять разделы ТЗ в виде приложений, вводить дополнительные, исключать или объединять подразделы ТЗ.

В ТЗ на части системы не включают разделы, дублирующие содержание разделов ТЗ на АС в целом.

2.3. В разделе «Общие сведения» указывают:

1) полное наименование системы и ее условное обозначение;

2) шифр темы или шифр (номер) договора;

3) наименование предприятий (объединений) разработчика и заказчика (пользователя) системы и их реквизиты;

4) перечень документов, на основании которых создается система, кем и когда утверждены эти документы;

5) плановые сроки начала и окончания работы по созданию системы;

6) сведения об источниках и порядке финансирования работ;

7) порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы.

2.4. Раздел «Назначение и цели создания (развития) системы» состоит из подразделов:

1) назначение системы;

2) цели создания системы.

2.4.1. В подразделе «Назначение системы» указывают вид автоматизируемой деятельности (управление, проектирование и т. п.) и перечень объектов автоматизации (объектов), на которых предполагается ее использовать.

Для АСУ дополнительно указывают перечень автоматизируемых органов (пунктов) управления и управляемых объектов.

2.4.2. В подразделе «Цели создания системы» приводят наименования и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автоматизации, которые должны быть достигнуты в результате создания АС, и указывают критерии оценки достижения целей создания системы.

2.5. В разделе «Характеристики объекта автоматизации» приводят:

1) краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию;

2) сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды.

Примечание: Для САПР в разделе дополнительно приводят основные параметры и характеристики объектов проектирования.

2.6. Раздел «Требования к системе» состоит из следующих подразделов:

1) требования к системе в целом;

2) требования к функциям (задачам), выполняемым системой;

3) требования к видам обеспечения.

Состав требований к системе, включаемых в данный раздел ТЗ на АС, устанавливают в зависимости от вида, назначения, специфических особенностей и условий функционирования конкретной системы. В каждом подразделе приводят ссылки на действующие НТД, определяющие требования к системам соответствующего вида.

2.6.1. В подразделе «Требования к системе в целом» указывают:

– требования к структуре и функционированию системы;

– требования к численности и квалификации персонала системы и режиму его работы;

– показатели назначения;

– требования к надежности;

– требования безопасности;

– требования к эргономике и технической эстетике;

– требования к транспортабельности для подвижных АС;

– требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы;

– требования к защите информации от несанкционированного доступа;

– требования по сохранности информации при авариях;

– требования к защите от влияния внешних воздействий;

– требования к патентной чистоте;

– требования по стандартизации и унификации;

– дополнительные требования.

6.3. Современные стандарты

Сегодня стандарты системной и программной инженерии (СиПИ) разрабатываются, как правило, в неразрывном единстве и являют развитую систему, в которой представлены словари, своды знаний, руководства по принципам описания систем и процессов, гармонизированные между собой основополагающие стандарты, со-

держащие описание ключевых процессов ЖЦ систем и программных средств, а также руководства по их применению, стандарты оценки качества процессов ЖЦ систем, оценки зрелости процессов и управления ИТ-сервисами. Кроме того, в последние 2–3 года появились стандарты, в которых определяются детализированные требования к отдельным процессам ЖЦ систем. Новейшие стандарты СиПИ относятся не только к вопросам управления ЖЦ, но и к другим актуальным разделам современной системной и программной инженерии. Совершенствуются методы разработки, и, соответственно, появляются новые стандарты, относящиеся к «традиционной» системной инженерии. Планируется, что ISO/IEC/JTC1 в сотрудничестве с такими ведущими организациями, как IEEE и INCOSE, в ближайшие годы сумеет гармонизировать между собой две эти группы стандартов СиПИ.

Рассмотрим кратко содержание двух наиболее популярных отечественных стандартов определяющих жизненный цикл систем, и согласованных с принятыми международными стандартами.

6.3.1. ГОСТ Р 57193–2016 – СиПИ. Процессы жизненного цикла. (Разработан на основе стандарта ISO/IEC 15288:2008 System and software engineering – Living cycle processes)

Стандарт устанавливает общую структуру для описания жизненного цикла систем, созданных людьми. Он определяет набор процессов и связанную с ними терминологию. Эти процессы могут применяться на любом уровне иерархии структуры системы. Выбранные наборы этих процессов могут применяться на протяжении всего жизненного цикла для управления и выполнения этапов жизненного цикла системы. Это достигается за счет участия всех заинтересованных сторон, с конечной целью достижения удовлетворенности клиентов. ISO/IEC 15288: 2008 также обеспечивает процессы, которые поддерживают определение, контроль и улучшение процессов жизненного цикла, используемых в организации или проекте. Организации и проекты могут использовать эти процессы жизненного цикла при приобретении и поставке систем. ISO/IEC 15288: 2008 относится к тем системам, которые созданы человеком и могут быть сконфигурированы с одним или несколькими следующим элементами: аппаратное обеспечение, программное обеспечение, данные, люди, процессы (например, процессы предоставления услуг пользователям), процедуры (например, инструкции оператора), средства, материалы и естественные объекты. Когда системный элемент является программным обеспече-

нием, для реализации этого элемента системы могут использоваться процессы жизненного цикла программного обеспечения, задокументированные в ISO/IEC 12207: 2008. ISO/IEC 15288: 2008 и ISO/IEC 12207: 2008 гармонизированы для одновременного использования по одному проекту или в одной организации.

Системные понятия

Системы, которые рассматриваются в настоящем стандарте, создаются человеком и используются для обеспечения продукцией и/ или услугами в определенной окружающей среде в интересах пользователей и других заинтересованных сторон. Системы могут формироваться из аппаратных средств, программных средств, данных, людей, процессов (например, процессов для оказания услуги пользователям), процедур (например, инструкций оператору), услуг, материалов и других естественно возникающих сущностей. Согласно предоставлениям пользователя, они рассматриваются как продукты или услуги.

Восприятие и определение специальной системы, ее архитектуры и элементов зависят от интересов и ответственностей заинтересованных сторон. Рассматриваемая система одной заинтересованной стороны может быть представлена как системный элемент другой заинтересованной стороны в рассматриваемой системе. Кроме того, рассматриваемая система может предоставляться другой заинтересованной стороной как часть окружающей среды для данной системы.

Следующие положения являются основными относительно характеристик рассматриваемой системы:

– определенные границы характеризуют значимые потребности и практические решения;

– существуют иерархические или иные отношения между системными элементами;

– какая-либо сущность на любом уровне в рассматриваемой системе может быть рассмотрена как система;

– система включает интегрированное, определенное множество нижестоящих системных элементов;

– свойства характеристик в границах системы определяются результатами взаимодействий между системными элементами;

– люди могут рассматриваться как пользователи, внешние к системе, и как системные элементы (т. е. операторы) в пределах системы;

– система может быть рассмотрена в изоляции как некая сущность, например, как продукт или набор функций, способных к взаимодействию с окружающей средой, т. е. как множество услуг.

Безотносительно границ, выбранных для определения системы, понятия в стандарте являются исходными и разрешают любому практику соответствовать или приспосабливать частные случаи жизненных циклов к его системным принципам.

Структура системы

Процессы жизненного цикла системы в стандарте описаны относительно системы, которая составлена из ряда системных элементов для взаимодействия, каждый из которых может быть реализован таким образом, чтобы выполнить соответствующие ему заданные требования.

Ответственность за реализацию любого системного элемента может быть делегирована другой стороне через соглашение.

Отношения между системой и ее полным множеством системных элементов могут представляться в иерархии до простейших элементов, относящихся к рассматриваемой системе. Для более сложной системы предполагаемый системный элемент должен быть самостоятельно рассмотрен как система (которая в свою очередь состоит из системных элементов), прежде чем полное множество системных элементов может быть с уверенностью определено. Таким же способом рекурсивно применяются к рассматриваемой системе и соответствующие процессы жизненного цикла системы, чтобы привести ее структуру к такому представлению, когда осознанные и управляемые системные элементы могут быть реализованы другой стороной (сделаны, закуплены или повторно используемы). В то время как часто в системах подразумеваются иерархические отношения, в действительности имеется большое число систем, которые по каким-то причинам не являются иерархическими, такие как, например, сети и иные распределенные системы.

Организации

Организацию или часть организации, которые заключает соглашение, называют стороной соглашения. Стороны могут относиться к одной организации или различным организациям. Организация может быть малой, например, состоять из одного человека, если он наделен ответственностью и полномочиями. В неофициальных определениях организация, которая ответственна за выполнение процесса, иногда упоминается с названием этого процесса. Например, организацию, выполняющую процесс приобретения, иногда называют приобретающей стороной. Другие примеры включают поставщика, конструк-

тора, сопровождающую сторону и оператора. В настоящем стандарте к организациям применяются также некоторые другие термины: «пользователь» может быть организацией, которая извлекает выгоду от использования продукта или услуги; «заказчик» обращается к пользователю и приобретающей стороне; «заинтересованная сторона» в конкретном проекте обращается к отдельному человеку или организации с каким-либо интересом.

Процессы и организации связаны только функционально. Настоящий стандарт не навязывает и не подразумевает какой-либо структуры для организации, и не определяет то, как специальные процессы должны выполняться специальными частями организации. Все это находится в рамках ответственности организации, которая применяет настоящий стандарт, чтобы определить подходящую структуру для организации и установить соответствующие роли для выполнения процессов. Процессы в настоящем стандарте формируют исчерпывающее множество для возможностей их использования различными организациями. Организация, малая или большая, в зависимости от ее бизнес-цели или стратегии приобретения, может выбрать соответствующее множество процессов (и соответствующих действий и задач), чтобы достичь своих целей. Организация может выполнять один или более процессов. Настоящий стандарт предназначен для применения внутри организации или вне ее, между двумя или более организациями. Когда стандарт применяется внутри организации, две соглашающихся стороны обычно действуют в соответствии с соглашением, которое может формально меняться при различных обстоятельствах. Когда стандарт применяется вне организации, две соглашающиеся стороны обычно действуют в соответствии с контрактом. Настоящий стандарт использует термин «соглашение», чтобы соответствовать любой ситуации. Любой проект согласно целям, описанным в настоящем стандарте, осуществляется в пределах контекста организации. Это важно, так как проект системы зависит от различных результатов, произведенных с использованием бизнес-процессов организации, например от работников для комплектации обслуживающего персонала проекта и оборудования. Для решения этих задач настоящий стандарт предоставляет множество процессов организационного обеспечения проекта. Важно отметить, что изначально и процессы организационного обеспечения проекта, и процессы проекта могут оказаться не вполне адекватными для оперирования бизнесом и использования в проекте. Вместе с тем процессы, рассматриваемые в совокупности, предназначены для установления минимального множества зависимостей проекта, размещаемого в организации.

Современный бизнес стремится разрабатывать устойчивое множество процессов жизненного цикла, которые неоднократно применяются к проектам. Поэтому настоящий стандарт предназначен для применения на любом уровне в организации или на уровне проекта. Организация может принять стандарт и добавить к его положениям соответствующие процедуры, методы, инструментарии и политику. Проект организации обычно более соответствует процессам организации, нежели непосредственно настоящему стандарту. В некоторых случаях проекты могут быть выполнены организацией, которая не имеет соответствующего множества процессов, принятых на уровне организации. Такой проект может применять условия настоящего стандарта непосредственно к самому себе (т. е. к проекту).

Модель жизненного цикла системы

Каждая система имеет жизненный цикл. Жизненный цикл может быть описан с использованием абстрактной функциональной модели, которая представляет собой осмысление потребностей в системе, ее реализации, эксплуатации, развитии и списании. Система развивается через свой жизненный цикл как результат действий, выполняемых и управляемых специалистами организации, используя для этих действий процессы. Детали в модели жизненного цикла выражены в терминах этих процессов, их результатов, отношений и последовательности. Настоящий стандарт не отдает предпочтений какой-либо специальной модели жизненного цикла. Вместо этого определяется множество процессов, которые названы процессами жизненного цикла. Все это может использоваться при определении жизненного цикла системы. Кроме того, настоящий стандарт не предписывает какой-либо специальной последовательности процессов в пределах модели жизненного цикла. Последовательность процессов определяется проектными целями и выбором модели жизненного цикла системы.

Стадии жизненного цикла системы

Жизненные циклы изменяются согласно природе, целям, использованию системы и преобладающим обстоятельствам. Каждая стадия имеет свои цели и вклад в жизненный цикл и рассматривается при планировании и осуществлении всего жизненного цикла системы. Стадии представляют собой главные периоды жизненного цикла, связанные с системой, и касаются состояния в описании системы или непосредственно состояния самой системы. Стадии описывают главное развитие и контрольные точки достижения системы в ее жизненном

цикле. Они дают развитие прохождениям решений по жизненному циклу. Эти прохождения решений используются организациями с тем, чтобы понимать и управлять изначальной неопределенностью и рисками, связанными с затратами, сроками и функциональными возможностями при создании или использовании системы. Тем самым стадии предоставляют организациям такую структуру, в пределах которой у руководства организации существует представление наивысшего уровня и возможности управления проектом и техническими процессами. Обычно типичные стадии жизненного цикла системы включают замысел, разработку, производство, применение, поддержку и выведение из эксплуатации. Организации используют стадии различным образом так, чтобы удовлетворить стратегиям разнообразного бизнеса и снизить риски. Использование стадий одновременно и в различных последовательностях может привести к формам жизненного цикла с отчетливо различными характеристиками. Дальнейшая проработка этих понятий может быть найдена в руководствах, посвященных применению управления жизненным циклом.

Понятия процесса

Определение процессов жизненного цикла в настоящем стандарте основано на трех принципах:

– для каждого процесса жизненного цикла вводятся строгие отношения между его результатами, действиями и задачами;

– зависимости между процессами стремятся уменьшить до максимально достижимой степени;

– любой из процессов должен обладать возможностями быть выполненным любой отдельной организацией в конкретном жизненном цикле.

Описание процессов

Каждый процесс настоящего стандарта описывается в терминах следующих атрибутов:

атрибут «Название» передает область процесса в целом;

атрибут «Цели» описывает цели выполнения процесса;

атрибут «Выход (выходные результаты)» выражает заметные результаты, ожидаемые от успешной работы процесса;

атрибут «Действия» – это множества связанных задач процесса;

атрибут «Задачи» – это требования, рекомендации или допустимые действия, используемые для поддержки достижения конкретных результатов. Дополнительные детали относительно формы описания процесса – в ГОСТ Р 57098–2016/ISO/ IEC TR 24774:2010.

Общие характеристики процессов

В дополнение к основным признакам, описанным в предыдущем пункте, процессы могут быть охарактеризованы другими атрибутами, общими для всех процессов. ГОСТ Р ИСО/МЭК 15504-2–2009 определяет общие атрибуты процесса, которые характеризуют шесть уровней достижения в пределах структуры измерений для обеспечения возможностей процесса. Приложение С включает термины атрибутов процесса, которые способствуют достижению высоких уровней возможности процесса, как это определено в ГОСТ Р ИСО/МЭК 15504-2–2009.

Процессы

Введение. Настоящий стандарт объединяет действия, которые могут быть выполнены в жизненном цикле системы, в четыре группы процессов:

1. Процессы соглашения.
2. Процессы организационного обеспечения проекта.
3. Процессы технического управления.
4. Технические процессы.

Список всех процессов приведен на рис. 6.1.

Каждый из процессов в пределах этих групп описан в терминах его цели и желаемых результатов, получаемых путем выполнения действий и решения задач для достижения этих результатов. Процессы, описанные в настоящем стандарте, не предназначены для устранения или препятствования использованию дополнительных процессов, которые организация находит полезными. Описание каждой группы процессов предоставлено в четырех подразделах, следующих ниже.

Помогая параллельному использованию настоящего стандарта и стандарта ISO/IEC/IEEE 12207:2015, соответствующие процессы, изложенные в данном разделе, имеют те же самые номера подразделов. Многие из процессов настоящего стандарта кажутся подобными процессам, свойственным программным средствам, но они сохраняют весомые различия, основанные на целях, результатах и потребителях. Пользователям настоящего стандарта и стандарта ISO/IEC/IEEE 12207:2015, следует убедиться в том, что для каждого определенного процесса ими учтены различные обоснования и примечания.

Процессы соглашения

Организации являются участниками создания и пользователями систем. Одна организация (действующая как приобретающая сторона)

может задать работу другой (действующей как поставщик) для поставки продуктов или услуг. Это достигается использованием соглашений. В общем случае организации действуют одновременно или последовательно и как приобретающие стороны, и как поставщики систем. Если приобретающая сторона и поставщик находятся в одной организации, то процессы соглашения могут использоваться с меньшим количеством формальностей.

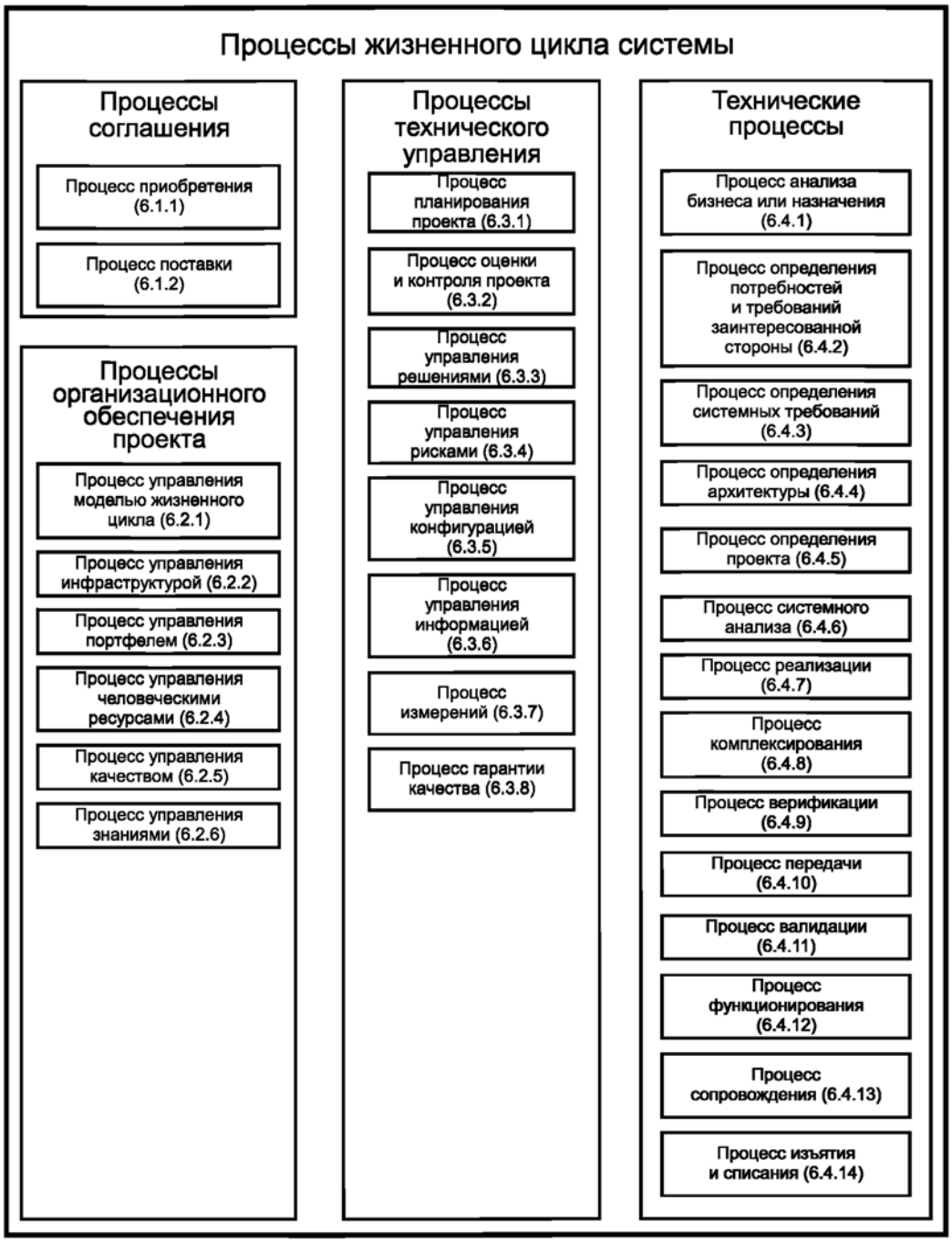


Рис. 6.1. Структура стандарта ГОСТ Р 57193–2016 – СиПИ. Процессы жизненного цикла

Точно так же, чтобы договориться о соответствующих ответственностях в организации, проекте и о технических функциях, процессы могут использоваться в пределах организации.

Процессы организационного обеспечения проекта

Процессы организационного обеспечения проекта (всего 6 процессов) сосредоточены на том, чтобы наличие необходимых ресурсов позволило проекту удовлетворить потребности и ожидания заинтересованных сторон. Процессы организационного обеспечения проекта обычно охватывают стратегический уровень управления и совершенствования бизнеса организации или обязательства с условием и развертыванием ресурсов и активов, и управлением рисками в конкурентоспособных или неопределенных ситуациях. Процессы организационного обеспечения проекта устанавливают окружающую среду для осуществления проекта.

Организация устанавливает процессы и модели жизненного цикла, которые будут использоваться проектами; устанавливает, перенаправляет или отменяет проекты; обеспечивает требуемыми ресурсами, включая человеческие и финансовые ресурсы; устанавливает и контролирует показатели качества для систем и других поставок, которые реализуются проектами в интересах внутренних и внешних заказчиков. Процессы организационного обеспечения проекта создают строгий деловой имидж для многих организаций и подразумевают коммерческую пользу и иные выгоды. Вместе с тем процессы организационного обеспечения проекта одинаково относятся и к некоммерческим организациям, так как они являются также ответственными перед заинтересованными сторонами за ресурсы и противодействие рискам в их обязательствах. Настоящий стандарт может применяться некоммерческими организациями в той же степени, что и коммерческими.

Процессы технического управления

Процессы технического управления (всего 8 процессов) сосредоточены на управлении ресурсами и активами, распределяемыми руководством организации с применением этих процессов, чтобы выполнить соглашения, в которых участвуют организация или несколько организаций. Они касаются технических усилий проектов, особенно планирования в терминах стоимости, сроков и достижений, проверки действий. Процессы осуществляются для обеспечения гарантий выполнения планов и критериев работы, определения

и выбора корректирующих действий, которые влияют на развитие и достижения. Они используются:

– для установления и выполнения технических планов относительно проекта;

– управления информацией с помощью технической команды;

– оценки технического продвижения согласно планам по продуктам системы или услугам;

– управления техническими задачами через их завершение;

– помощи в принятии решений.

Примечание. Техническое управление является применением технических и административных ресурсов для планирования, организации и управления техническими функциями).

Обычно в любой организации сосуществуют несколько проектов. Процессы технического управления могут использоваться на корпоративном уровне.

Технические процессы

Технические процессы (всего 14 процессов) сосредоточены на технических действиях по всему жизненному циклу. Они преобразуют потребности заинтересованных сторон сначала в продукт и затем, применяя этот продукт, оказывают для достижения удовлетворенности заказчика устойчивые услуги, когда и где это необходимо. Технические процессы применяются в заказах по созданию и использованию системы, проявляется ли это в форме некоторой модели или является готовым изделием. Процессы применяются на любом уровне в иерархии структуры системы. Для подробного рассмотрения всех процессов необходимо обратиться к стандарту.

6.3.2. ГОСТ Р ИСО/МЭК 12207–2010. Информационная технология. Системная и программная инженерия. Процессы Жизненного цикла программных средств

Данный стандарт заменяет стандарт ISO/IEC 12207: 2008 – System and software engineering – Software living cycle processes и устанавливает общую структуру процессов жизненного цикла программного обеспечения с четко определенной терминологией, на которую может ссылаться индустрия программного обеспечения.

ISO/IEC 12207: 2010 применяется к приобретению систем, программных продуктов и услуг, а также к поставке, разработке, эксплуатации, обслуживанию и удалению программных продуктов и программной части системы, независимо от того, выполняются они

внутри или вне организации. Включены те аспекты определения системы, которые необходимы для обеспечения контекста программных продуктов и услуг. ISO/IEC 12207: 2008 также обеспечивает процесс, который может быть использован для определения, контроля и улучшения процессов жизненного цикла программного обеспечения. Процессы, действия и задачи ISO/IEC 12207: 2008 выполняются либо самостоятельно, либо совместно с ISO/IEC 15288, а также могут применяться при приобретении системы, содержащей программное обеспечение.

Настоящий стандарт разработан для сторон, приобретающих системы, программные продукты и услуги, а также для поставщиков, разработчиков, операторов, сопровожденцев, менеджеров (в том числе, менеджеров по качеству) и пользователей программных продуктов.

Настоящий стандарт предназначен для использования при двусторонних отношениях и может применяться также в случае, когда обе стороны принадлежат одной и той же организации. Такие отношения могут варьироваться от неформального соглашения вплоть до официального контракта. Настоящий стандарт может использоваться одной из сторон через самостоятельно выбираемую совокупность процессов, что не исключает применения настоящего стандарта поставщиками или разработчиками готовых программных продуктов.

Ограничения

В настоящем стандарте не детализируются процессы жизненного цикла в терминах методов или процедур, необходимых для удовлетворения требований и достижения результатов процесса.

Настоящий стандарт не устанавливает требований к документации в части ее наименований, форматов, определенного содержания и носителей для записи. Настоящий стандарт может потребовать разработки документов подобного класса или типа, например, различных планов. Настоящий стандарт, однако, не предусматривает, чтобы такие документы разрабатывались или комплектовались раздельно или каким-то образом объединялись. Эти решения остаются за пользователем настоящего стандарта.

Примечание. В стандарте ISO/IEC 15289 установлены требования к информационному содержанию разделов документации, описывающей процессы жизненного цикла.

Настоящий стандарт не устанавливает конкретной модели жизненного цикла системы или программных средств, разработки методологии, методов, моделей или технических приемов. Сто-

роны, применяющие настоящий стандарт, отвечают за выбор модели жизненного цикла для программных проектов и отображение процессов, действий и задач, представленных в настоящем стандарте, на эту модель. Стороны также ответственны за выбор и применение методов разработки программных средств и за выполнение действий и задач, подходящих для программного проекта.

Настоящий стандарт не должен противоречить политикам, процедурам и нормам применяющей его организации, национальным законам и регулирующим документам. Каждое такое противоречие должно быть разрешено до начала использования настоящего стандарта.

В общем случае настоящий стандарт применяется как для программных продуктов, так и для программных услуг. Условия конкретных процессов определяют их применимость.

Отношения между системами и программными средствами

Настоящий стандарт устанавливает строгую связь между системой и применяемыми в ней программными средствами. Такая связь основывается на общих принципах системной инженерии. Программное средство трактуется как единая часть общей системы, выполняющая определенные функции в данной системе, что осуществляется посредством выделения требований к программным средствам из требований к системе, проектирования, производства программных средств и объединения их в систему. Этот принцип является фундаментальной предпосылкой для настоящего стандарта, в котором программные средства всегда существуют в контексте системы, даже если система состоит из единственного процессора, выполняющего программы. В таком случае программный продукт или услуга всегда рассматриваются как одна из составных частей системы. Например, в настоящем стандарте проводится различие между анализом системных требований и анализом требований к программным средствам, так как в общем случае построение системной архитектуры определяет системные требования для различных составных частей системы, а анализ требований к программным средствам предопределяет требования к ним, исходя из системных требований, назначенных каждой программной составной части. Конечно, в некоторых случаях непрограммных элементов в системе может быть настолько мало, что можно не делать различия между анализом системы и анализом программных средств.

Настоящий стандарт имеет сильную взаимосвязь с ИСО/МЭК 15288 (см. рассмотренный выше ГОСТ Р 57193–2016 – СиПИ) и может

использоваться вместе с ним. Во многих случаях процессы в настоящем стандарте непосредственно соответствуют процессам в этом стандарте, но с некоторой спецификой для программных продуктов и услуг. Примечательным примером является то, что процесс реализации программных средств в настоящем стандарте является специальным, частным случаем процесса реализации, приведенного в ИСО/МЭК 15288.

В случае, если система имеет важные непрограммные элементы, то организация может по желанию применять ИСО/МЭК 15288 для обеспечения соответствующих процессов жизненного цикла. Для каждого программного элемента системы организации следует применять процесс реализации программных средств из настоящего стандарта для создания программного элемента.

В случае, если настоящий стандарт применяется в сочетании с ИСО/МЭК 15288, то должно учитываться любое незначительное несоответствие в терминологии. В ИСО/МЭК 15288 производится декомпозиция системы на совокупность системных «элементов». Некоторые из этих элементов могут определяться как программные продукты, которые реализуются с использованием настоящего стандарта. В настоящем стандарте применяется термин «составная часть» для указания на некоторый основной элемент системы. Короче говоря, в настоящем стандарте используется термин «составная часть» тогда, когда в ИСО/МЭК 15288 используется термин «элемент программного средства».

Некоторые составные части могут, в конечном счете, назначаться как объект менеджмента конфигурации; тогда они называются «составными частями конфигурации». Процесс проектирования архитектуры программных средств преобразует составные части в «компоненты», а процесс детального проектирования программных средств переводит «компоненты» в «программные блоки».

Процессы в настоящем стандарте составляют полную совокупность для охвата различных организаций. Организация (малая или крупная) в зависимости от ее деловых целей или стратегии приобретения может выбрать подходящую совокупность процессов (а также связанных с ними действий и задач) для выполнения этих целей. Организация может выполнять один или несколько процессов. Например, по условиям контракта или применения настоящего стандарта конкретная сторона не должна выполнять ни процесс приобретения, ни процесс поставки, но она может выполнять другие процессы. Процесс может реализовываться одной или несколькими организа-

циями. Примером процесса, выполняемого более чем одной организацией, является процесс ревизии программных средств.

Настоящий стандарт предназначен для применения двумя или большим числом организаций (как внутри, так и вне организаций). Если он применяется внутри организации, то две стороны соглашения обычно действуют согласно положениям, установленным в соглашении, которые могут изменяться с соблюдением принятых правил при различных обстоятельствах. Если он применяется при отношениях с внешними сторонами, то стороны соглашения обычно действуют согласно положениям, изложенным в контракте. Для того чтобы облегчить применение настоящего стандарта как для внутренних целей, так и для контрактных отношений, задачи выражаются на языке контракта. Если стандарт применяется для внутренних целей, то положения контракта следует интерпретировать как установленную в пределах организации исполнительскую дисциплину.

Для целей настоящего стандарта предполагается, что любой проект осуществляется в пределах контекста организации. Это важно, так как программный проект зависит от различных результатов, производимых деловыми процессами организации, например, найма работников для укомплектования штатного персонала проекта и средств обеспечения проекта. Для этой цели настоящий стандарт предоставляет совокупность процессов «организационного обеспечения проекта». Предполагается, что эти процессы не охватывают ни деловой деятельности, ни какого-либо отдельного процесса проекта. Вместо этого процессы, рассматриваемые в совокупности, предназначаются для установления минимального множества зависимостей, которые проект возлагает на организацию.

Общий список процессов, рассматриваемых в данном стандарте, приведен на рис. 6.2.

Описание процессов

Процессы в настоящем стандарте описываются способом, подобным способу, представленному в ИСО/МЭК 15288, для того чтобы обеспечить использование обоих стандартов в одной организации или проекте.

Все процессы разделяются на три множества:

– процессы в контексте системы, которые включают следующие группы процессов;

– процессы соглашения (2 процесса);

– процессы организационного обеспечения проекта (5 процессов);

– процессы проекта (7 процессов);

– технические процессы (11 процессов);
– специальные процессы программных средств;
– процессы реализации программных средств (7 процессов);
– процессы поддержки программных средств (8 процессов);
– процессы повторного применения программных средств (3 процесса);
– процессы проектирования доменов;
– процессы менеджмента повторного применения программ;
– процессы менеджмента повторного применения активов.

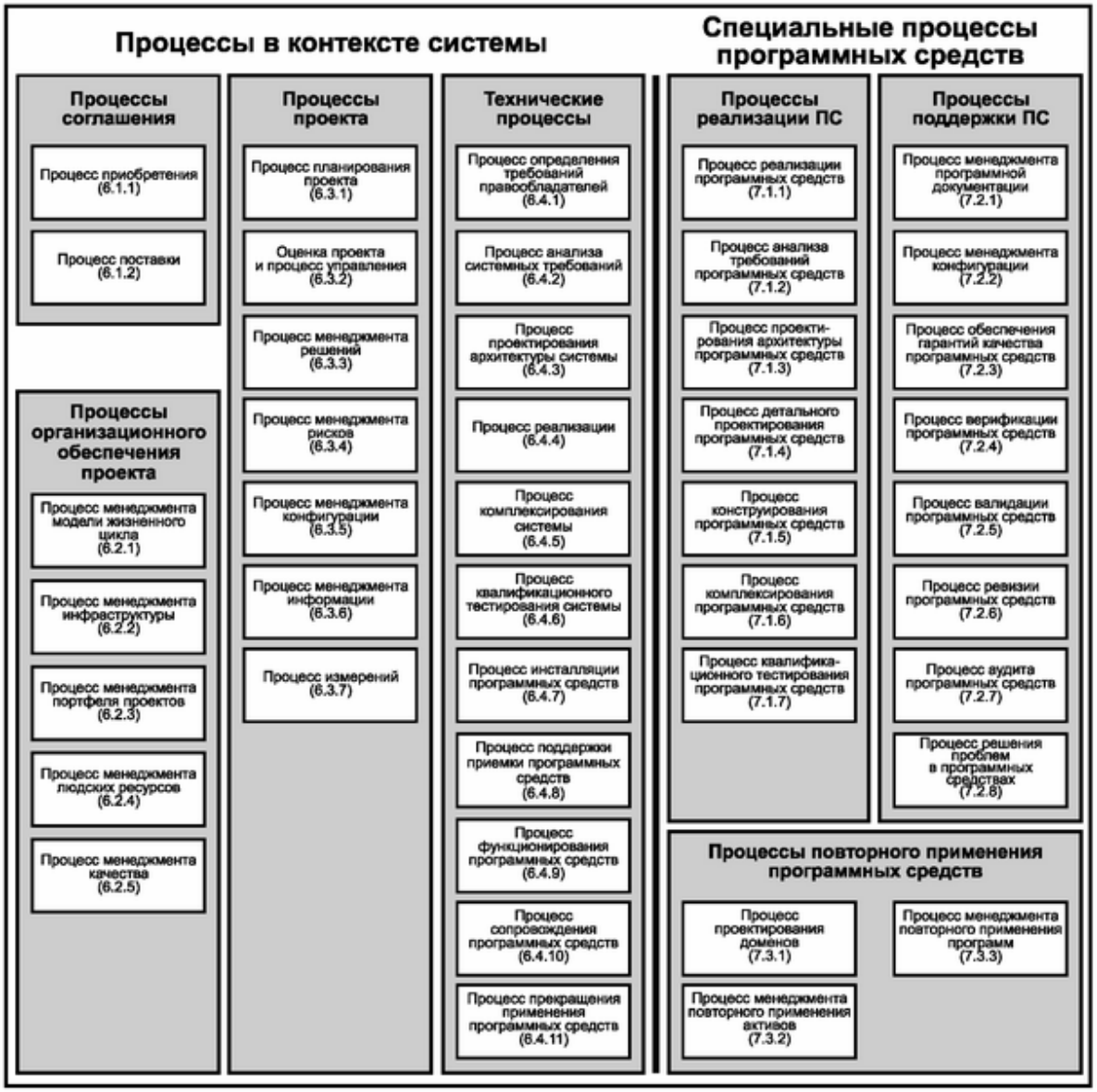


Рис. 6.2. ГОСТ Р ИСО/МЭК 12207–2010. Информационная технология. Системная и программная инженерия. Процессы Жизненного цикла программных средств

Каждый процесс настоящего стандарта описывается в терминах следующих атрибутов:

– наименование – передает область применения процесса как целого;

– цель – описывает конечные цели выполнения процесса;

– выходы – представляют собой наблюдаемые результаты, ожидаемые при успешном выполнении процесса;

– деятельность – является перечнем действий, используемых для достижения выходов;

– задачи – представляют собой требования, рекомендации или допустимые действия, предназначенные для поддержки достижения выходов процесса.

Для подробного рассмотрения всех процессов необходимо обратиться к стандарту.

Контрольные вопросы и задания

1. Охарактеризуйте различие уровней стандартизации: международные, национальные, отраслевые, корпоративные, отдельной организации.

2. В чем сходство и различие понятий компетентности в образовательных стандартах и трудовые функции в профессиональных стандартах?

3. Как различить обобщенную трудовую функцию и трудовую функцию в профессиональных стандартах?

4. Какие модели с жизненного цикла ПО вы знаете? Назовите их достоинства, недостатки и область применения.

5. Сравните стиль требований в стандартах ГОСТ 34.601–90 и ГОСТ Р ИСО/МЭК 12207.

6. Ознакомьтесь с профессиональными стандартами:

06.003 «Архитектор программного обеспечения»,

06.017 «Руководитель разработки программного обеспечения»,

06.028 «Системный программист».

7. Составьте список трудовых функций для одной из обобщенных трудовых функций выбранного стандарта, относящихся к 4 уровню квалификации. Проанализируйте эти трудовые функции.

ЗАКЛЮЧЕНИЕ

Программная инженерия в настоящее время обрела устоявшуюся методологию и проверенную технологию. Она позволяет успешно создавать множество сложных программных систем в самых разных областях человеческой деятельности.

Как пишет Гради Буч, потребность в сложных программных системах растет в мире с ошеломляющей быстротой. По мере того, как увеличивается производительность компьютеров и все больше людей узнают об их возможностях, возникает желание автоматизировать все более сложные процессы. Фундаментальная ценность программной инженерии как устоявшейся технологии заключается в том, что она позволяет человечеству сосредоточиться на решении творческих задач при создании сложных систем.

В последние годы темпы развития информационных технологий настолько ускорились и приобрели настолько глобальный характер, что можно говорить о наступлении новой цифровой эры в человеческой истории. Наша страна не остается в стороне от этого процесса, о чем свидетельствует принятие в 2017 году уже упомянутой Стратегии развития информационного общества в Российской Федерации на 2017–2030 годы, утвержденной указом Президента России от 9 мая 2017 года № 203 (www.pravo.gov.ru).

Стратегия определяет цели, задачи и меры по реализации внутренней и внешней политики Российской Федерации в сфере применения информационных и коммуникационных технологий, направленные на развитие информационного общества, формирование национальной цифровой экономики, обеспечение национальных интересов и реализацию стратегических национальных приоритетов. Несомненно, что реализация Стратегии даст толчок и развитию программной инженерии.

Данное пособие, конечно, и в малой степени не способно осветить все аспекты современной программной инженерии, оно представляет лишь краткий обзор этой области знаний и технологий.

Более или менее полные учебники по программной инженерии содержат многие сотни страниц. Например, цикл из 8 учебников по программной инженерии В. В. Липаева содержит в сумме 2424 страницы.

Поэтому будущему специалисту в этой области предстоит освоить огромный объем теоретических знаний и практических навыков и непрерывно пополнять свои знания, учитывая бурное развитие программной инженерии.

Для этого требуется участие в практической работе по созданию больших программных систем – в процессе обучения, а главное – при работе по приобретенной в вузе профессии.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Гуд, Г. Х. Системотехника. Введение в проектирование больших систем / Г. Х. Гуд, Р. Э Макол. – М. : Советское радио, 1962. – 383 с.

2. Моисеев, Н. Н. Математические задачи системного анализа / Н. Н. Моисеев – М. : Наука. 1981. – 488 с.

3. Антонов, А. В. Системный анализ : учебник для вузов / А. В. Антонов – М. : Высш. шк., 2004. – 454 с.

4. Советов, Б. Я. Моделирование систем / Б. Я. Советов, С. А. Яковлев. – М. : Высш. шк., 1995. – 350 с.

5. Сиротинина, Н. Ю. История и методология информатики и вычислительной техники : учеб. пособие / Н. Ю. Сиротинина. – Томск : Изд-во «СПБ Графикс», 2012. – 196 с.

6. Плотинский, Ю. М. Теоретические и эмпирические модели социальных процессов : учеб. пособие / Ю. М. Плотинский. – М. : Логос, 1988. – 280 с.

7. Сухомлин, В. А. Введение в анализ информационных технологий : учебник для вузов / В. А. Сухомлин. – М. : Горячая линия – Телеком, 2003. – 427 с.

8. Доррер, Г. А. Теория информационных процессов и систем : учеб. пособие / Г. А. Доррер ; СибГТУ. – Красноярск, 2010. – 242 с.

9. Брукс, Ф. Мифический человеко-месяц, или Как создаются программные системы / Ф. Брукс. – М. : Символ-Плюс, 2010. – 304 с.

10. Липаев, В. А. Программная инженерия. Основы методологии / В. А. Липаев – М. : Теис, 2006 – 609 с.

11. Батоврин, В. К. Толковый словарь по системной и программной инженерии : учеб. пособие / В. К. Батоврин. – М. : ДМК Пресс, 2012. – 280 с.

12. Орлик, С. В. Введение в программную инженерию и управление жизненным циклом ПО [Электронный ресурс] / С. В. Орлик. – Электрон. дан. – Режим доступа: https://vk.com/sistemnaya_inzheneriya_levenchuk. – Загл. с экрана.

13. Лаврищева, Е. М. Технология программирования и программная инженерия : учебник / Е. М. Лаврищева – М. : Юрайт, 2017. – 432 с.

14. Мацяшек, Л. А. Практическая программная инженерия на основе учебного примера : пер. с англ. / Л. А. Мацяшек, Б. Л. Лонг. – М. : Бином. Лаборатория знаний, 2009. – 956 с.

15. Буч, Г. Язык UML. Руководство пользователя : пер. с англ. / Г. Буч, Д. Рамбо, Д. Джекобсон. – М. : ДМК, 2000. – 432 с.

16. Боэм, Б. Инженерное проектирование программного обеспечения / Б. Боэм. – М. : Радио и связь, 1985. – 512 с.

17. Vliet, H. van Software Engineering: Principles and Practice / H. van Vliet. – 2nd ed. – John Wiley & Sons, 2000.

18. Карпенко, С. Н. Программная инженерия: назначение, основные принципы и понятия / С. Н. Карпенко. – Н. Новгород : ННГУ, 2005. – 103 с.

19. Checland, P. B. Models Validation in Soft Systems Practice / P. B. Checland // System Research. – 1995. – Vol. 12, № 1. – P. 47–54.

20. Software Engineering Methodologies for Embedded Systems [Электронный ресурс]. – Электрон. дан. – Режим доступа: www.mooseproject.org. – Загл. с экрана.

21. Управление требованиями [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://docplayer.ru/47033385-Upravlenie-trebovaniyami.html https://habr.com/ru/post/245625/. – Загл. с экрана.

24. Royce, W. W. Managing the Development of Large Software Systems [Электронный ресурс] / W. W. Royce. – 1970. – Электрон. дан. – Режим доступа: http://facweb.cti.depaul.edu/jhuang/is553/Royce.pdf. – Загл. с экрана.

25. Boehm Barry. A Spiral Model of Software Development and Enhancement [Электронный ресурс] / Boehm Barry // IEEE Computer. – 1988. – Vol. 21, № 5. – P. 61–72. – Электрон. дан. – Режим доступа: www.computer.org/computer/homepage/misc/Boehm/r5061.pdf. – Загл. с экрана.

24. Кодекс этики программной инженерии [Электронный ресурс] // IEEE-CS/ACM Software Engineering Ethics and Professional Practices. – Электрон. дан. – Режим доступа: http://www.computer.org/tab/seprof/code.htm#Public. – Загл. с экрана.

25. Guide to the Systems Engineering Body of Knowledge (SEBoK) [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://www.sebokwiki.org/wiki/. – Загл. с экрана.

26. Guide to the Software Engineering Body of Knowledge (SWEBOK) Version 3.0 // Международный стандарт ISO/IEC TR 19759 от 2015 г.

Учебное издание

Доррер Георгий Алексеевич

МЕТОДОЛОГИЯ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Учебное пособие

Редактор *П. С. Бороздов*
Оригинал-макет и верстка *М. А. Светлаковой*

Подписано в печать 15.06.2021. Формат 60×84/16. Бумага офисная.
Печать плоская. Усл. печ. л. 11,0. Уч.-изд. л. 13,0. Тираж 50 экз.
Заказ . С 135/21.

Санитарно-эпидемиологическое заключение
№ 24.49.04.953.П.000032.01.03 от 29.01.2003 г.

Редакционно-издательский отдел СибГУ им. М. Ф. Решетнева.
660037, г. Красноярск, просп. им. газ. «Красноярский рабочий», 31.
E-mail: rio@mail.sibsau.ru. Тел. (391) 201-50-99.

Отпечатано в редакционно-издательском центре
СибГУ им. М. Ф. Решетнева.
660049, г. Красноярск, просп. Мира, 82. Тел. (391) 227-69-90.