Министерство науки и высшего образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

ФАКУЛЬТЕТ ДИСТАНЦИОННОГО ОБУЧЕНИЯ (ФДО)

Ю. П. Ехлаков

ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Учебное пособие

Томск
2019

УДК 004.41(075.8)
ББК 32.973-018я73
Е 934

Рецензенты:

В. Ф. Тарасенко, д-р техн. наук, профессор кафедры теоретической кибернетики Национального исследовательского Томского государственного университета;

О. И. Жуковский, канд. техн. наук, доцент кафедры автоматизации обработки информации Томского государственного университета систем управления и радиоэлектроники

Ехлаков, Ю. П.

Е 934 Основы программной инженерии : учебное пособие / Ю. П. Ехлаков. – Томск : Эль Контент, 2019. – 128 с.

ISBN 978-5-4332-0280-1

В учебном пособии описана модель технологического процесса промышленного производства программного продукта. Приводятся материалы об отечественных и зарубежных стандартах, регламентирующих процессы жизненного цикла программных продуктов; моделях разработки программных продуктов; перечне и содержании этапов и областей знаний управления программными проектами; выводе на рынок и коммерциализации программных продуктов.

Для студентов, обучающихся по направлению подготовки 09.03.04 «Программная инженерия» (уровень бакалавриата).

ISBN 978-5-4332-0280-1

Оглавление

Введение

Традиционный подход к программированию как к искусству создания уникальных программ профессионалами-одиночками уходит в прошлое. В настоящее время производство и продажа программных продуктов (ПП) приобрели черты рентабельного, но подверженного большим рискам вида бизнеса. Это связано, с одной стороны, с низкой материалоемкостью процессов производства и высокой долей интеллектуального труда создателей программ, а с другой – с тем, что ПП, являясь результатом творческого труда, не поддается точному оцениванию ни по времени создания, ни по требуемому бюджету и поэтому создается в условиях повышенного риска.

Все бизнес-процессы разработки ПП должны иметь черты современного промышленного производства с эффективной организацией труда команды программистов, регламентацией и структуризацией технологии создания программных продуктов. При этом жесткая конкуренция разработчиков в сфере IT-сектора экономики объективно требует от компании обращать должное внимание как на качество создаваемых продуктов, так и на экономику и маркетинг их разработки и продвижения на рынок. Кроме того, промышленные способы производства должны основываться на современных стандартах и инструментальных средствах создания программных продуктов, поддерживающих все этапы его жизненного цикла. В круг интересов программной инженерии как методологии промышленного создания ПП входят вопросы выявления и анализа требований пользователей, проектирования, конструирования, тестирования, управления выпуском, поставки, внедрения и сопровождения качественных и экономичных программных продуктов, снабженных технической документацией.

Содержание учебного процесса, порядок и условия его проведения определяются (регламентируются) Федеральным государственным образовательным стандартом высшего образования – бакалавриат по направлению подготовки 09.03.04 «Программная инженерия» (ФГОС3++) [1], а также серией профессиональных стандартов: в сфере «Связь, инфокоммуникационные и коммуникационные технологии», соответствующих направлению подготовки 09.03.04 «Программная инженерия» (уровень бакалавриата), разрабатываемых IT-компаниями, входящими в Ассоциацию предприятий компьютерных и информационных технологий [2] и утверждаемых соответствующими приказами Министерства труда и социальной защиты Российской Федерации.

В федеральном государственном образовательном стандарте изложены обязательные требования при реализации вузом профессиональной образовательной программы, перечислены виды, сферы и задачи профессиональной деятельности выпускников, описаны универсальные и общепрофессиональные компетенции, которые должны быть сформированы у выпускника при освоении основной профессиональной образовательной программы, а также рекомендации по выбору профессиональных стандартов, на основе которого вуз может самостоятельно определять набор профессиональных компетенций.

Для формирования профессиональных компетенций основной профессиональной образовательной программы по направлению подготовки 09.03.04 «Программная инженерия» (уровень бакалавриата) вузам рекомендуется использовать профессиональные стандарты: «Программист», «Специалист по тестированию в области информационных технологий», «Системный аналитик», «Системный программист». В этих документах перечислены цели профессиональной деятельности, обобщенные и «элементарные» трудовые функции и действия, которые выпускник обязан выполнять при занятии соответствующей должности, а также требования к уровню образования и обучения.

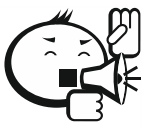
Профессиональная деятельность *программиста* заключается в разработке требований и проектировании архитектуры программного продукта; разработке и отладке программного кода; проверке работоспособности и рефакторинге исходного кода; интеграции программных модулей и компонентов; верификации выпусков программного продукта. Профессиональная деятельность *системного аналитика* ориентирована на разработку, восстановление и сопровождение требований к программному продукту на протяжении его жизненного цикла, в том числе: разработку и сопровождение требований к отдельным функциям системы; создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности; концептуальное, функциональное и логическое проектирование систем среднего и крупного масштаба и сложности. Профессиональная деятельность *специалиста по тестированию в области информационных технологий* связана с оценкой качества разрабатываемого программного продукта путем проверки соответствия продукта заявленным требованиям, сбора и передачи информации о несоответствиях. Основными видами профессиональной деятельности *системного программиста* являются разработка, отладка, модификация и поддержка системного программного обеспечения, включая разработку, отладку, модификацию и

поддержку системного программного обеспечения и систем управления базами данных.

Освоение образовательной программы данного направления подготовки позволит выпускнику:

- использовать методы математического моделирования для описания, анализа и проектирования бизнес-процессов в различных предметных областях, проводить экспериментальные исследования для их оптимизации;
- разрабатывать алгоритмы и программные продукты, пригодные для практического использования, применять современные информационные технологии и программные средства, в том числе отечественного производства при проектировании, конструировании и тестировании программных продуктов;
- инсталлировать программное и аппаратное обеспечение для информационных и автоматизированных систем с учетом основных требований информационной безопасности;
- участвовать в разработке технической документации, связанной с эксплуатацией и сопровождением программных продуктов.

Программная инженерия как профессия имеет определенные обязательства перед обществом: продукты, созданные программистами, могут оказать как позитивное, так и негативное влияние на жизнь и деятельность пользователей. Очевидно, что разработчики программного обеспечения должны соблюдать определенную этику и профессионализм в своей деятельности. Кодекс этики программной инженерии определяет мораль, правила и нормы поведения профессионалов в области программной инженерии, их обязательства и ответственность по отношению к обществу и друг к другу. Кодекс состоит из преамбулы и совокупности принципов, которых должен придерживаться профессионал [3].



Преамбула к Кодексу этических норм профессионала в области программной инженерии содержит следующие принципы:

1. Вследствие специфики своих ролей в процессе создания программного обеспечения инженеры имеют неограниченные возможности приносить пользу или причинять вред как самостоятельно, так и способствуя другим либо влияя на других.
2. Инженеры должны принять на себя обязательство сделать программную инженерию полезной и уважаемой профессией, чтобы быть уверенными в том, что их работа используется во благо.

В кодексе задекларировано восемь принципов в следующих сферах:

- согласование профессиональной деятельности с интересами общества;
- взаимоотношение между пользователем, работодателем и исполнителем разработки;
- достижение соответствия качества программного продукта лучшим профессиональным стандартам;
- соблюдение честности и независимости при профессиональных оценках качества процессов и продуктов;
- соблюдение этических норм в менеджменте и в сопровождении разработок;
- поддержка становления профессии в соответствии с кодексом этики;
- соблюдение этических норм во взаимоотношениях между коллегами;
- постоянное совершенствование программной инженерии как области профессиональной деятельности.

Целью изучения дисциплины «Основы программной инженерии» является формирование у студента осознания социальной значимости будущей профессии, мотивации к получению профессиональных знаний, пониманию и освоению основных методологий промышленного проектирования программных продуктов, отечественных и зарубежных стандартов, регламентирующих содержание и качество процессов жизненного цикла разработки ПП, интегрированных сред разработки программных продуктов и использование их на благо общества.

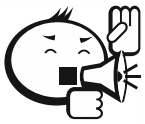
В учебном пособии речь пойдет о таких понятиях, как программное обеспечение (ПО), программная система (ПС), программный продукт (ПП), комплекс программ (КП). В общем случае программное обеспечение необходимо рассматривать как совокупность программных средств, предназначенных для решения

тех или иных задач, а программную систему, программный продукт, комплекс программ – как конкретизацию содержания ПО, имеющую конкретное название, торговую марку, снабженную документацией.

Соглашения, принятые в учебном пособии

Для улучшения восприятия материала в данном учебном пособии используются пиктограммы и специальное выделение важной информации.

Эта пиктограмма означает определение или новое понятие.



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

Пример

Эта пиктограмма означает пример. В данном блоке автор может привести практический пример для пояснения и разбора основных моментов, отраженных в теоретическом материале.

Контрольные вопросы по главе

1 Основные понятия программной инженерии как промышленной технологии создания программных продуктов

1.1 Определение и особенности программного продукта

В настоящее время IT-компании, связанные с производством программных продуктов (ПП), так или иначе, выбирают одну из двух бизнес-моделей деятельности: разработку и продвижение собственных программных продуктов (продуктовая, или тиражная, модель) или разработку уникального ПП «под заказ» (заказная модель). Понятие программного продукта, предназначенного для практического использования, можно раскрыть в виде следующего определения.

Совокупность записанных на носителях данных программных компонентов, являющихся продуктом промышленного производства, предназначенных для практического использования путем поставки, передачи или продажи пользователю, снабженных технической документацией, рекламными материалами, инструкциями по обучению пользователей, гарантийными обязательствами по сопровождению и обслуживанию.

Безусловно, применение каждой из бизнес-моделей таит в себе свои риски. При использовании заказной модели имеется риск разработать «под заказ» прикладной программный продукт, работающий с ошибками, непригодный для сопровождения и модификации. Кроме того, возможен риск «затянуть» проект или попасть в слишком опасную зависимость от постоянно меняющихся требований заказчика и т. д.

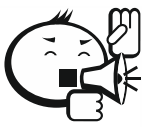
Использование продуктовой модели предполагает наличие востребованного на рынке (в том числе и глобальном) продукта (или портфеля продуктов) и обеспечение его тиражирования. С точки зрения оценки бизнеса компании-разработчика продуктовая модель более перспективна, в силу того что сама компания является непосредственным производителем новых проектов и технологий. При этом малыми ресурсами могут быть созданы инновационные рыночные продукты, имеющие большой экономический и коммерческий потенциал. Это, в

свою очередь, существенно улучшает условия, при которых могут быть получены инвестиции в случае капитализации компании. Использование продуктовой модели разработки ПП также требует пересмотра организационных процессов деятельности компании – от управления программным проектом к управлению программным продуктом как объектом экономических отношений на рынке.

Программный продукт, созданный по продуктовой модели, может поставляться пользователям как в виде продажи лицензий, так и в виде услуги. В первом случае речь идет о поставке, развертывании и внедрении ПП на программно-технических средствах заказчика по договору (контракту) купли-продажи, при этом имущественное право на ПП переходит к заказчику. Во втором случае программный продукт поставляется пользователям по SaaS-модели (англ. *software as a service* – программное обеспечение как услуга; англ. *software on demand* – программное обеспечение по требованию) в виде сервиса, как одной из форм облачных вычислений. При этом заказчику предоставляется доступ к готовому ПП, развернутому на программно-технических средствах разработчика. Основное преимущество модели SaaS для потребителя услуги состоит в том, что заказчики платят не за владение программным обеспечением как таковым, а за его аренду.

Таким образом, в отличие от классической схемы от продажи лицензий программного обеспечения заказчик несет сравнительно небольшие периодические затраты, и ему не требуется инвестировать значительные средства в приобретение прикладной программы и необходимых программно-платформенных и аппаратных средств для его развертывания, а затем поддерживать его работоспособность. С точки зрения разработчика модель SaaS позволяет эффективно бороться с нелегальным использованием программного обеспечения, поскольку оно как таковое не попадает к заказчику. Кроме того, модель SaaS позволяет уменьшить затраты на развертывание и внедрение систем технической поддержки продукта и пользователей.

В условиях рыночной экономики программные продукты, являясь объектом авторских прав, выступают в виде принципиально нового инновационного продукта, вовлечение которого в хозяйственный оборот происходит в процессе коммерциализации (купли-продажи, переуступки прав собственности) и капитализации (постановки на баланс, инвестирования в уставной капитал).



Таким образом, программный продукт представляет собой интеллектуальный цифровой товар и обладает рядом специфических особенностей, которые можно разделить на две группы [4]:

1) характеристики ПП как объекта промышленного производства, предназначенного для продажи:
   - ПП представляет собой публикацию текста программы/ программ на языке программирования или в виде исполняемого кода, зафиксированного на материальном носителе (компьютере, дисковом накопителе и др.), который может быть продан или передан, при этом обладание материальным носителем не делает его владельца уникальным собственником;
   - создание ПП связано с постоянными изменениями функционала, сроков разработки и затрат, обусловленных отсутствием у потенциальных потребителей четко сформулированных требований к продукту и слабым представлением о технологии его использования в практической деятельности;
   - необходимость адаптации стандартов на процессы жизненного цикла (ЖЦ) программного продукта к конкретным условиям ввиду того, что в существующих документах по регламентации данного вида деятельности процессы ЖЦ разработки ПП описаны в общем виде и прямо не ориентированы на специфику создаваемого продукта;
   - в структуре стоимости ПП относительно невысоки затраты на его изготовление (тиражирование), что обусловлено низкой стоимостью производственных операций по созданию копий и высокой стоимостью разработки ПП, в которой основную часть составляют затраты на оплату труда относительно небольшой группы специалистов;
   - ПП создается в условиях повышенного риска, поскольку, являясь результатом творческого труда, не поддается точному оцениванию ни по времени создания, ни по требуемому бюджету;
   - вовлечение ПП в хозяйственный оборот происходит в процессе его коммерциализации (купли-продажи, переуступки

прав собственности) и капитализации (постановки на баланс, инвестирования в уставный капитал);

2) характеристики ПП как объекта интеллектуальной собственности:
   - нематериальная природа существования ПП;
   - ПП может обмениваться, но при этом не происходит его полного отчуждения;
   - ПП может быть неоднократно продан, при этом одновременно выступать объектом нескольких рыночных сделок;
   - не исчезает и не изнашивается в процессе использования.

. . . . . .

1.2 Модель технологического процесса создания программного продукта

Программную инженерию следует рассматривать в трех взаимосвязанных аспектах:

1) как регламентную, соответствующую с принятым стандартам последовательность действий, операций, работ по получению программных продуктов;
2) совокупность методологий и инструментальных средств выполнения действий, операций, работ при создании программных продуктов;
3) описание организационных регламентов деятельности команды разработчиков при создании программных продуктов.

Процесс промышленной разработки программного обеспечения представляет собой специфический технологический процесс преобразования исходных требований заказчика в готовый программный продукт (рис. 1.1).

Создание программного продукта протекает в соответствии с принятыми на предприятии *внешними стандартами* и нормативными документами, регламентирующими содержание и качество процессов жизненного цикла разработки ПП. К таким документам, в частности, относятся:

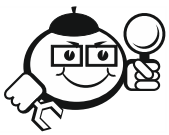
- ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»;
- IEEE-1074–1997 «Процессы и действия жизненного цикла программного обеспечения» (Developing software life cycle processes);

- Единая система программной документации (ЕСПД): ГОСТ 19.102–77 ЕСПД «Стадии разработки»;
- ГОСТ Р ИСО/МЭК 15910–2002 «Процесс создания документации пользователя программного средства»;
- ГОСТ Р ИСО 9127–94 «Документация пользователя и информация на упаковке для потребительских программных пакетов»;
- ГОСТ Р ИСО/МЭК 25040–2014 «Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Процесс оценки»;
- CMM – Capability Maturity Model (Модель зрелости процесса конструирования ПО) и т. д.

Рис. 1.1 – Модель технологического процесса создания программного продукта

На вход технологического процесса создания программного продукта поступают *сведения о предметной области.*

В формализованном виде предметная область описывается в виде упорядоченного набора бизнес-процессов, адекватно отображающих деятельность предприятия-заказчика по преобразованию исходных ресурсов в готовую продукцию.



Пример

Пример последовательности бизнес-процессов получения материального продукта представлен на рисунке 1.2. Бизнес-процессы преобразования исходного сырья материалов комплектующих в готовую материальную продукцию регламентируются *внутренними стандартами предприятия*: соответствующей технической документацией, положениями о деятельности структурных подразделений, должностными инструкциями сотрудников.

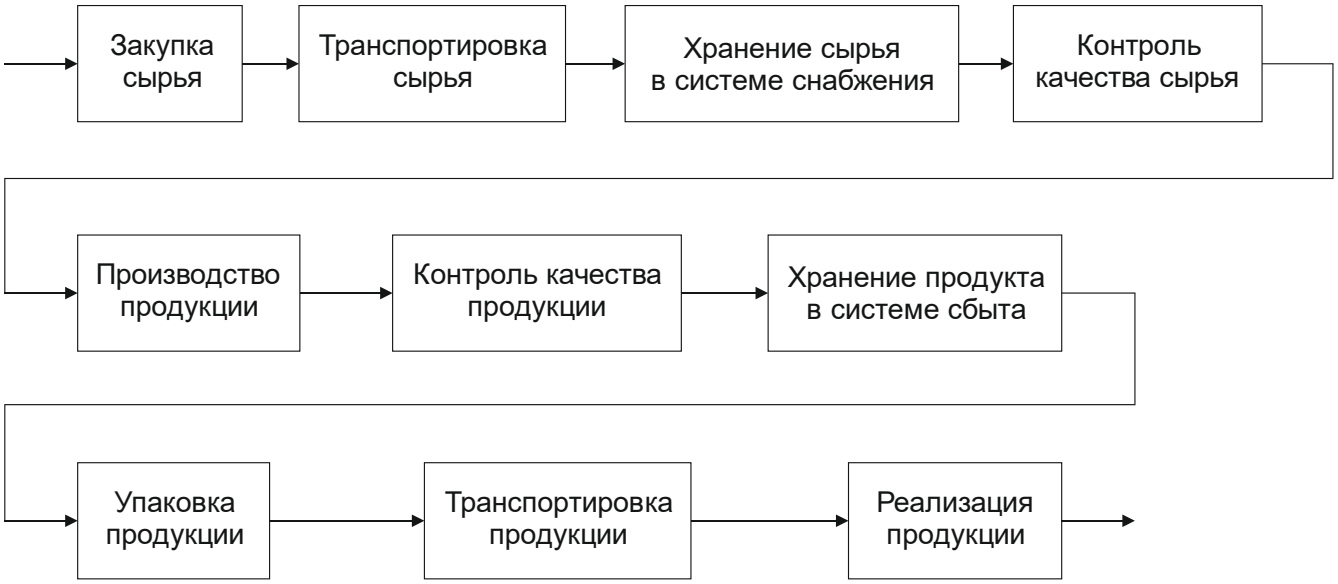
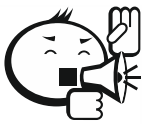


Рис. 1.2 – Бизнес-процессы создания материального продукта

Основным инструментом анализа и совершенствования бизнес-процессов является моделирование. Для построения *моделей бизнес-процессов* используются, как правило, структурные и объектно-ориентированные подходы.

В основе структурных методов моделирования бизнеса лежит декомпозиция системы на подсистемы, которые, в свою очередь, делятся на более мелкие составляющие и т. д. Базовые принципы структурного подхода: 1) «разделяй и

властвуй» – принцип решения сложных проблем путем их разбиения на множество мелких задач, легких для понимания и решения; 2) иерархическое упорядочивание – принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.



Наибольшее распространение получили следующие методы структурного моделирования: IDEF0 – функциональные модели, основанные на методе структурного анализа и проектирования SADT (Structured Analysis and Design Technique) Дугласа Росса; IDEF1X – модели данных, основанные на диаграммах «сущность – связь» (ERD, Entity-Relationship Diagrams); IDEF3 – диаграммы потоков работ (Work Flow Diagrams); DFD (Data Flow Diagrams) – диаграммы потоков данных.

Главным структурообразующим элементом в объектно-ориентированном подходе является объект. При моделировании бизнеса объектами прежде всего являются участники бизнес-процесса (*активные объекты*) – организационные единицы, конкретные исполнители, информационные системы, а также *пассивные объекты* – материалы, документы, оборудование, над которыми выполняют действия активные объекты. Таким образом, в объектно-ориентированном подходе модель бизнес-процессов строится вокруг участников процессов и их действий. Общепризнанным стандартом в области объектно-ориентированного подхода является язык моделирования UML. Более подробно методологии моделирования бизнес-процессов будут рассмотрены в главе 3.

Процессы жизненного цикла разработки программных продуктов регламентируются соответствующими стандартами. В соответствии с рекомендациями ГОСТ Р ИСО/МЭК 12207–2010 под жизненным циклом создания программного продукта следует понимать упорядоченную совокупность фаз (стадий, процессов, работ), охватывающих эволюционное изменение программного продукта с момента возникновения потребности в нем либо идеи его создания до полного изъятия ПП из эксплуатации.

На стадии *формулирования и спецификации требований* основная задача заключается в определении:

- бизнес-требований, отражающих финансовые, рыночные или другие показатели коммерческого характера, которые заказчики собираются получить от использования продукта;

- пользовательских требований, которые должны описывать задачи (возможности), которые программный продукт позволит решить пользователям в рамках своих служебных обязанностей (должностных инструкций);
- функциональных требований, раскрывающих функциональные возможности ПП, методы передачи и преобразования входных данных в результаты, условия и среду выполнения функций (например, защита и доступ к базам данных), интерфейсы к внутренним компонентам ПП и внешним приложениям;
- нефункциональных требований, регламентирующих характеристики качества программного продукта (функционал, надежность, эффективность, удобство эксплуатации и т. д.).

Стадия *проектирования* рассматривается как деятельность, результат которой содержит две составные части: архитектурный, или высокоуровневый, дизайн – описание высокоуровневой структуры организации компонентов системы, в том числе внутренних и внешних интерфейсов; детализированную архитектуру – описание каждого компонента в объеме, необходимом для конструирования.

Стади*я конструирования* заключается в разработке исполняемых программных модулей посредством комбинации кодирования, верификации, модульного тестирования, интеграционного тестирования и отладки; разработке технической документации.

Стадия *рыночного тестирования и релиза* (выпуска) программного продукта осуществляется в два этапа. *Альфа-тестирование* (внутреннее тестирование) – этап начала тестирования ПП специалистами-тестерами. *Бета-тестирование* (публичное тестирование) – привлечение потенциальных пользователей продукта для апробации ПП. После устранения выявленных в процессе тестирования недочетов осуществляется релиз – выпуск окончательной версии ПП, готового для использования и тиражирования.

Стадия *внедрения* заключается в инсталляции программного обеспечения на технических средствах заказчика, обучения пользователей, проведения опытной эксплуатации системы.

Стадия *ввода в промышленную эксплуатацию и сопровождения* начинается с первой продажи, установки и документального подтверждения начала использования программного продукта. Стадия с*опровождения* продолжается до

полного изъятия продукта из эксплуатации, что связано со спецификой использования программного продукта.

На выходе технологического процесса создания программного продукта получается программный продукт, снабженный технической документацией, рекламными материалами, инструкциями по обучению пользователей, гарантийными обязательствами по сопровождению и обслуживанию. ГОСТ 19.102–77 «Единая система программной документации» определяет следующие виды технической документации: ведомость эксплуатационных документов, описание применения, исходный текст ПП, руководство системного программиста, руководство программиста, руководство пользователя, руководство по техническому обслуживанию.

Сложные программные проекты не могут быть реализованы индивидуально программистом-универсалом, *разработка ведется командой разноплановых специалистов*. При этом на каждом из этапов жизненного цикла создания программного продукта участвуют IT-специалисты, выполняющие определенную последовательность действий, операций, работ. Введение специализации, распределение функциональных обязанностей и ответственности между членами команды проекта в зависимости от их квалификации приводит к тому, что специалисты, владеющие однотипными компетенциями, работают над проектом в одной функциональной группе.

В соответствии с методологией Microsoft Solutions Framework в команде проекта рекомендуется выделять следующие функциональные ролевые группы [5]:

- группа управления проектом;
- группа проектирования архитектуры;
- группа разработки программного продукта;
- группа тестирования;
- группа управления выпуском;
- группа обеспечения связи с заказчиком;
- группа управления продуктом.

В каждой из функциональных групп над проектом могут работать различные IT-специалисты: менеджер проекта, архитектор, бизнес-аналитик, разработчик-программист, специалист по тестированию, риск-менеджер, менеджер по работе с заказчиками.

CASE-средства представляют собой набор инструментальных средств моделирования, проектирования и разработки ПП, позволяющих в наглядной

форме моделировать предметную область, анализировать эту модель на всех этапах создания и сопровождения ПП, разрабатывать приложения в соответствии с потребностями пользователей. В основу большинства CASE-средств положены методологии структурного или объектно-ориентированного анализа и проектирования.

Объективная потребность в использовании CASE-средств определяется следующими особенностями программных проектов: сложность описания бизнес-процессов (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов; наличие взаимосвязанных компонентов ПП, имеющих свои локальные задачи, необходимость интеграции существующих и вновь разрабатываемых приложений; эксплуатация ПП в неоднородной среде на нескольких аппаратных платформах; участие в проекте отдельных групп разработчиков, различных по уровню квалификации и со своими сложившимся традициям использования тех или иных инструментальных средств; временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ПП.

По функциональному назначению можно выделить следующие типы CASE-средств:

- средства анализа и проектирования ПП, позволяют формировать статическую модель предметной области (прежде всего функциональную модель), например, в виде диаграмм функциональной декомпозиции или диаграмм потоков данных, обеспечивающих разработчикам возможность описывать архитектурный дизайн ПП, спецификации компонентов и интерфейсов алгоритмы и структур данных (схем баз данных);
- средства разработки приложений, предназначены для генерация программного кода на различных языках верхнего уровня;
- средства проектирования баз данных, обеспечивают процессы моделирования данных и генерацию схем баз данных (как правило, на языке SQL);
- средства реинжиниринга, позволяют проводить анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций;

- средства конфигурационного управления, обеспечивают управляемость и контролируемость процессов разработки и сопровождения ПП;
- средства тестирования, обеспечивают проверку соответствия приложения предъявляемым бизнес-требованиям или проверку и оценку производительности приложений;
- средства документирования, предназначены для автоматизации разработки проектной документации на всех фазах ЖЦ ПО;
- средства управления проектом, предназначены для планирования хода выполнения проекта, а также для сопровождения проекта (контроля и корректировки планов выполнения работ).

Грамотное использования CASE-технологии в индустриальном проектировании ПП обеспечивает: улучшение качества ПП за счет автоматического контроля и генерации отдельных элементов; возможность повторного использования компонентов разработки; повышение уровня адаптивности и качества сопровождения ИС; использование методологии прототипного проектирования; ускорение работы за счет автоматизированной генерации кода и автоматизированного документирования проекта; возможность коллективной разработки ИС в режиме реального времени.

Контрольные вопросы по главе 1

1. Приведите ключевые фразы в определении программного продукта, предназначенного для практического использования.
2. Прокомментируйте специфику продуктовой бизнес-модели разработки программных продуктов.
3. Прокомментируйте специфику заказной бизнес-модели разработки программных продуктов.
4. Перечислите специфику разработки программных продуктов как услуги.
5. Перечислите и прокомментируйте характеристики ПП как объекта промышленного производства, предназначенного для продажи.
6. Перечислите и прокомментируйте характеристики ПП как объекта интеллектуальной собственности.
7. Раскройте содержание модели технологического процесса создания программного продукта.

8. Какие элементы модели технологического процесса регламентируются внешними и внутренними стандартами?
9. Перечислите и прокомментируйте этапы жизненного цикла разработки программных продуктов.
10. Раскройте содержание структурного (функционального) и объектно-ориентированного подходов при описании бизнес-процессов предметной области.
11. Дайте понятие CASE-средств и перечислите их функциональные возможности.
12. Поясните понятия объективной потребности и эффективности при использовании CASE-средств.
13. Перечислите и прокомментируйте основные принципы Кодекса этических норм профессионала в области программной инженерии.

2 Моделирование бизнес-процессов предметной области

2.1 Структурный подход к построению моделей бизнес-процессов

Предметная область представляет собой набор бизнес-процессов, адекватно описывающих деятельность организации по производству определенных конечных продуктов и/или оказанию услуг.

В свою очередь под бизнес-процессом понимается:

- *совокупность взаимосвязанных и взаимодействующих видов деятельности, преобразующих входы в выходы, представляющие ценность для клиента;*
- *множество внутренних упорядоченных видов деятельности организации по преобразованию исходных ресурсов в готовую продукцию* (*услугу*).

Первым шагом по выявлению бизнес-процессов должно стать выделение основных продуктов (услуг) и выстраивание процессов в соответствии с жизненным циклом производства продуктов и/или оказанию услуг.

Жизненный цикл – строго упорядоченная совокупность процессов, описывающих эволюционное преобразование исходных ресурсов в конечные продукты и услуги.

Примеры моделей жизненного цикла разработки программного продукта и производства материального продукта представлены соответственно на рисунках 1.1 и 1.2.

Модель бизнес-процесса – это описание реальных процессов в определенных символах, которое дает представление о том, каким образом происходит преобразование входных ресурсов в продукцию (услуги), и позволяет дать оценку эффективности функционирования бизнеса. Такая модель получила название «*Как есть*». Совершенствование (реинжиниринг) бизнес-процессов модели «*Как есть*» либо проектирование нового бизнеса позволяют смоделировать представление о том, *как должен* функционировать бизнес, чтобы достигались поставленные цели. Новой модели присваивается название «*Как должно быть*».

Для измерения эффективности бизнес-процессов, планирования и контроля хода выполнения процесса необходимо определить *ключевые показатели результативности* (*метрики*) – измеримые характеристики (атрибуты), по которым можно судить, насколько эффективно выполняется процесс. Выбор метрик зависит от конкретного процесса. Например, для оценки бизнес-процессов разработки ПП это время, стоимость и качество.

При описании бизнес-процессов предметной области используются, как правило, структурная и объектно-ориентированная методологии проектирования.

Типичными представителями *структурной методологии* являются:

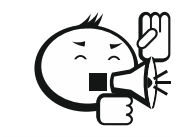
- методология построения диаграмм функционального моделирования – *методология IDEF0*;
- методология построения диаграмм *сущность* – связь – *диаграмма* «*сущность – связь*» (ERD);
- методология построения диаграмм потоков данных – *диаграмма потоков данных DFD.*

Методология IDEF0 позволяет описывать модели бизнес-процессов в различных предметных областях, включая разработку программного обеспечения, бизнес-анализ, проектирование, планирование и управление производственными системами, управление финансами и материально-техническими ресурсами.

Для отражения информации о конкретной системе в методология IDEF0 используется специальный графический язык описания диаграмм и фрагментов текста. На диаграммах все функции системы и их взаимодействия представлены как блоки (функции) и дуги (отношения).

Основной конструкцией модели является функциональный блок, представленный в виде прямоугольника и отображающий некоторую функцию (действие, процесс, операцию). Внутри блока записывается его наименование. Оно должно содержать глагол или отглагольное существительное. Например: «разработать проект», «изготовление продукта», «планирование».

Дуги, изображаемые на диаграмме в виде линий со стрелками на конце, играют роль связей блоков с внешней для них средой. Каждая из дуг имеет метку, характеризующую ее.



Назначение дуг зависит от стороны блока, в которую стрелка входит или выходит (рис. 2.1) [6]:

- «вход» (I – input) – дуги, входящие слева от блока. Они представляют собой предметы или данные, необходимые для выполнения функции блока (сырье, материалы, исходная информация);
- «выход» (O – output) – дуги, выходящие справа из блока. Они показывают предметы или данные, полученные в результате выполнения функции (продукция, услуга, выходные данные);
- «управление» (C – control) – дуги, входящие сверху блока. Они описывают условия или данные, которые управляют выполнением функции (инструкции, требования, стандарты);
- «механизм» (M – mechanism) – дуги, входящие снизу блока. Они обозначают исполнителей или средства, выполняющие функцию (персонал, подразделения фирмы, оборудование, инструменты, информационная система).

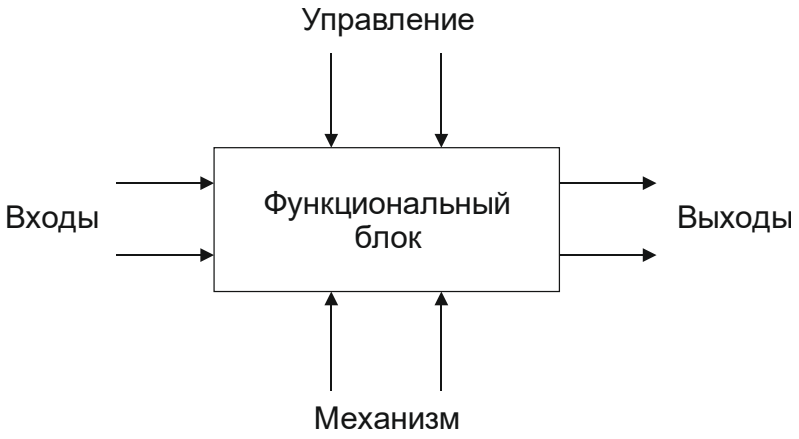


Рис. 2.1 – Функциональный блок IDEF0-диаграммы

Выход и вход показывают, что и из чего делается функциональным блоком, управление показывает, как и почему это делается, а механизм показывает, кем и с помощью чего это делается.

Функциональный блок может быть декомпозирован, т. е. представлен в виде совокупности других взаимосвязанных функциональных блоков, которые детально описывают исходный блок. Таким образом, IDEF0-модель состоит из набора иерархически связанных диаграмм (рис. 2.2). На диаграмме корневого уровня представлена вся система в виде одного блока и дуг, изображающих связи

с внешним окружением. На диаграмме декомпозиции первого уровня система представлена более детально в виде совокупности блоков-подмодулей, соединенных дугами друг с другом и с окружением. На диаграммах декомпозиции следующего уровня детализируются блоки диаграммы первого уровня и т. д.

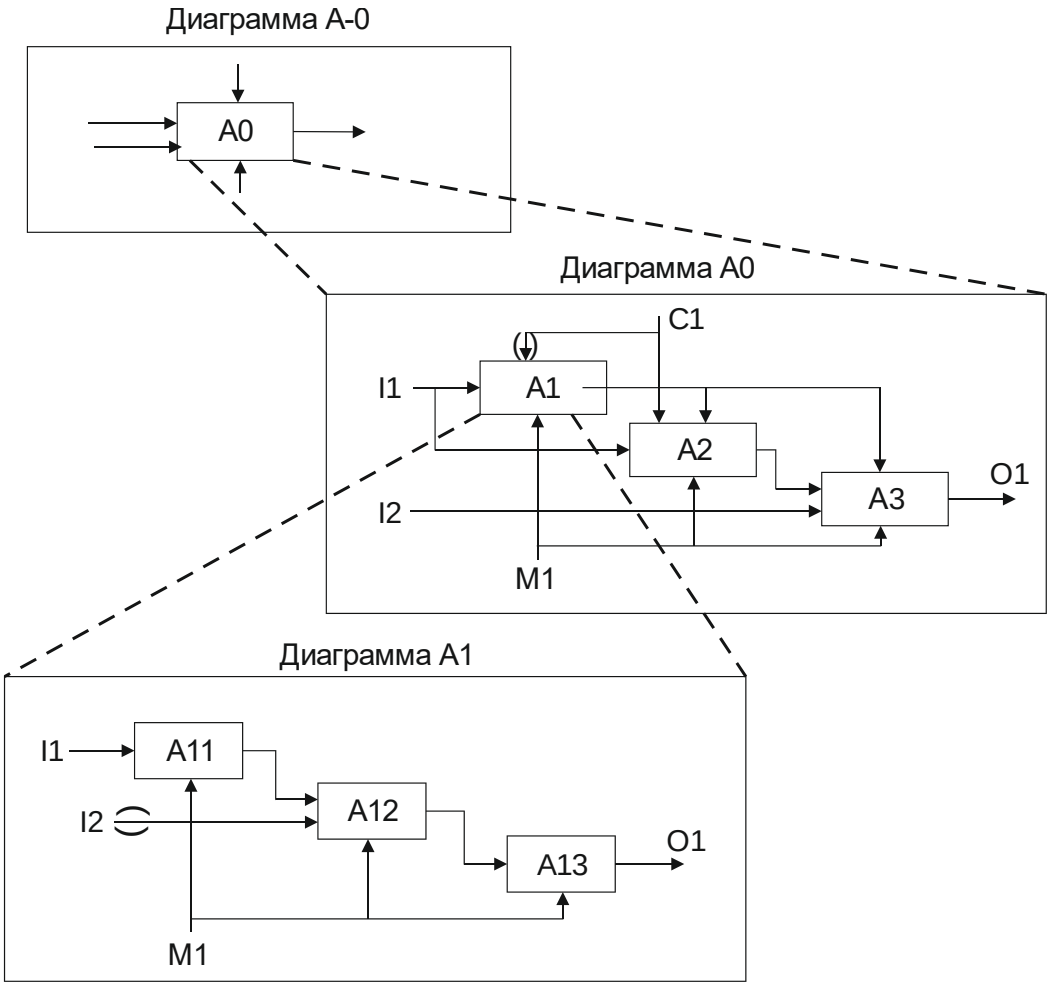


Рис. 2.2 – Иерархия диаграмм IDEF0-модели

Пример декомпозиции бизнес-процесса «Создание продукта» представлен на рисунках 2.3 и 2.4.

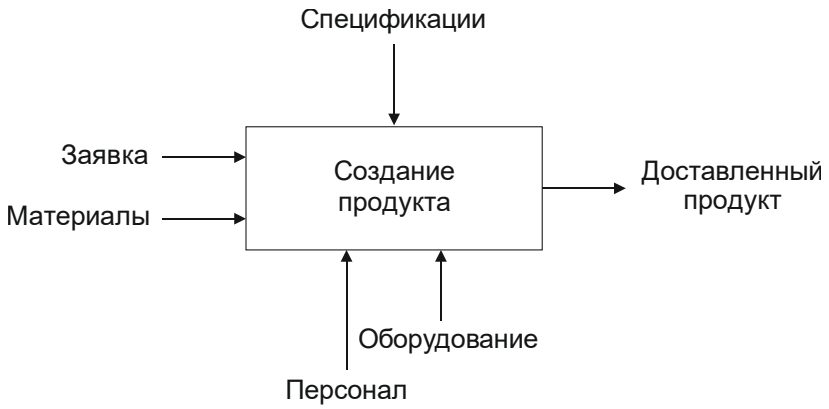


Рис. 2.3 – Пример контекстной диаграммы

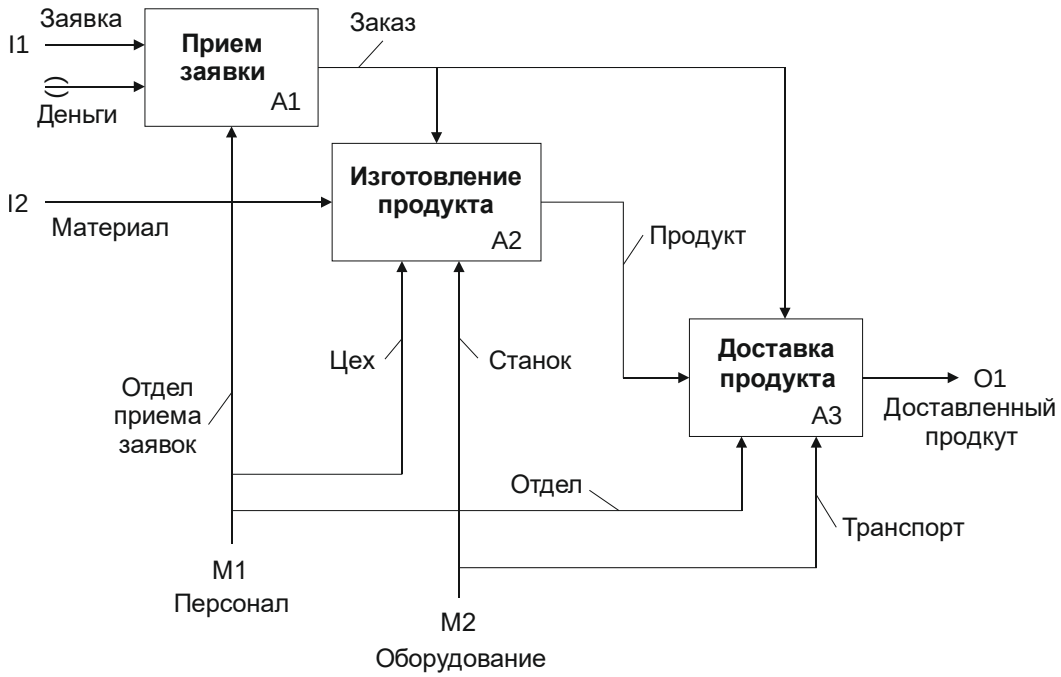


Рис. 2.4 – Пример диаграммы декомпозиции

Диаграмма «сущность – связь» (*ERD*) предназначена для графического представления моделей данных разрабатываемой программной системы и содержит некоторый набор стандартных обозначений для определения данных и отношений между ними. Основными понятиями (элементами) данной нотации являются сущности (entity) и связи (relationship). Для графического представления сущностей используются специальные обозначения (рис. 2.5) [6].

Рис. 2.5 – Пример диаграммы «сущность – связь»

Сущность *представляет произвольное множество реальных или абстрактных объектов, каждый из которых обладает одинаковыми свойствами и характеристиками. Каждый рассматриваемый объект может являться экземпляром одной и только одной сущности. Он должен иметь уникальное имя или идентификатор, а также отличаться от других экземпляров данной сущности.*

Связь *определяется как отношение или некоторая ассоциация между отдельными сущностями. Результатом моделирования является концептуальная модель предметной области, представленная в виде набора диаграмм. Современный инструментарий располагает сервисом создания физической модели базы данных, автоматически генерируемой на основе концептуальной модели.*

Диаграммы потоков данных DFD позволяют наглядно описать процессы обработки информации. С их помощью система разбивается на функциональные компоненты (процессы, которые преобразуют входные данные в выходные) и представляется в виде сети, связанной потоками данных. В методологии DFD используются четыре типа структурных элементов: внешние сущности, процессы, потоки данных, хранилища данных.

Внешние сущности *определяют элементы вне контекста системы, которые участвуют в процессе обмена информацией с системой, являясь источниками или приемниками информации. Внешние сущности изображают входы в систему и/или выходы из системы. Как правило, они представляют собой материальный предмет, физическое или юридическое лицо.*

Процессы *обозначают функции, операции, действия, которые описывают, каким образом входные потоки данных преобразуются в выходные. Процесс обозначается в виде прямоугольника со скругленными углами, разделенного на три поля. Верхнее поле содержит номер процесса, среднее – его имя, нижнее – имя исполнителя процесса.*

Потоки *данных используются для отображения взаимодействия процессов с внешней средой и между собой. Поток данных соединяет выход процесса с входом другого процесса и обозначается в виде именованной стрелки* (*имя отражает содержимое потока*).

Хранилища *данных представляют собой собственно данные, к которым осуществляется доступ. Эти данные также могут быть созданы или изменены процессами. В отличие от потоков данных, описывающих данные в движении, хранилища данных отображают данные в покое, т. е. данные, которые сохраняются в базе между последующими процессами.*

. . . . . .

Диаграммы потоков данных имеют несколько вариантов графического представления элементов структуры программной системы и ее интерфейсов. Элементы DFD-диаграммы и пример приведены соответственно на рисунках 2.6. и 2.7 [6]. Внешние сущности предлагается отображать в виде прямоугольных элементов; процессы преобразования данных обозначаются окружностями или овалами; потоки данных – стрелками с названиями; хранилища данных – отрезками горизонтальных параллельных линий.

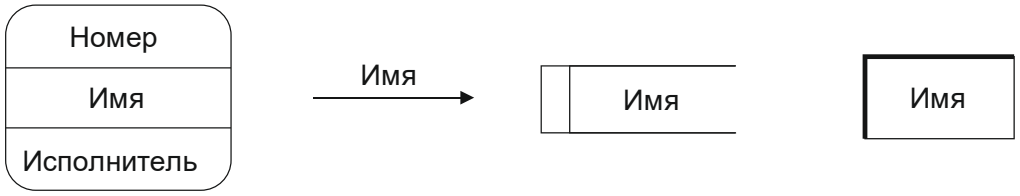


Рис. 2.6 – Элементы DFD-диаграммы в нотации Гейна – Сарсона:
а – процесс; *б* – поток данных; *в* – хранилище данных; *г* – внешняя сущность

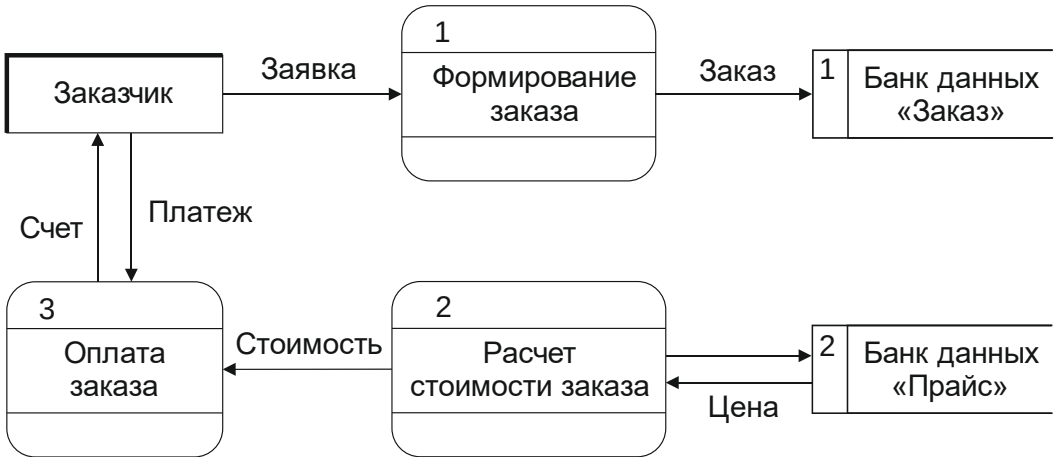


Рис. 2.7 – Пример DFD-диаграммы

2.2 Объектно-ориентированная методология построения моделей бизнес-процессов

Главным структурообразующим элементом в объектно-ориентированной методологии является объект. При моделировании бизнеса объектами являются прежде всего участники бизнес-процесса (активные объекты) – организационные единицы, конкретные исполнители, информационные системы, а также пассивные объекты – материалы, документы, оборудование, над которыми выполняют действия активные объекты. Таким образом, в объектно-ориентированном подходе модель бизнес-процессов строится вокруг участников процессов и их действий. Общепризнанным стандартом в области объектно-ориентированной методологии является язык моделирования UML. Моделирование бизнеса с помощью UML предполагает последовательное построение двух видов моделей:

1) прецедентной модели (аналога модели поведения), описывающей функциональность – бизнес-процессы (прецеденты) и их взаимодействие с окружением;
2) объектной модели (аналога структурной модели), описывающей внутреннее устройство бизнеса – объекты, участвующие в выполнении бизнес-процессов и их взаимодействие.

Пример

Интегрированная модель предметной области в нотации UML представляется в виде совокупности диаграмм (рис. 2.8) [6].

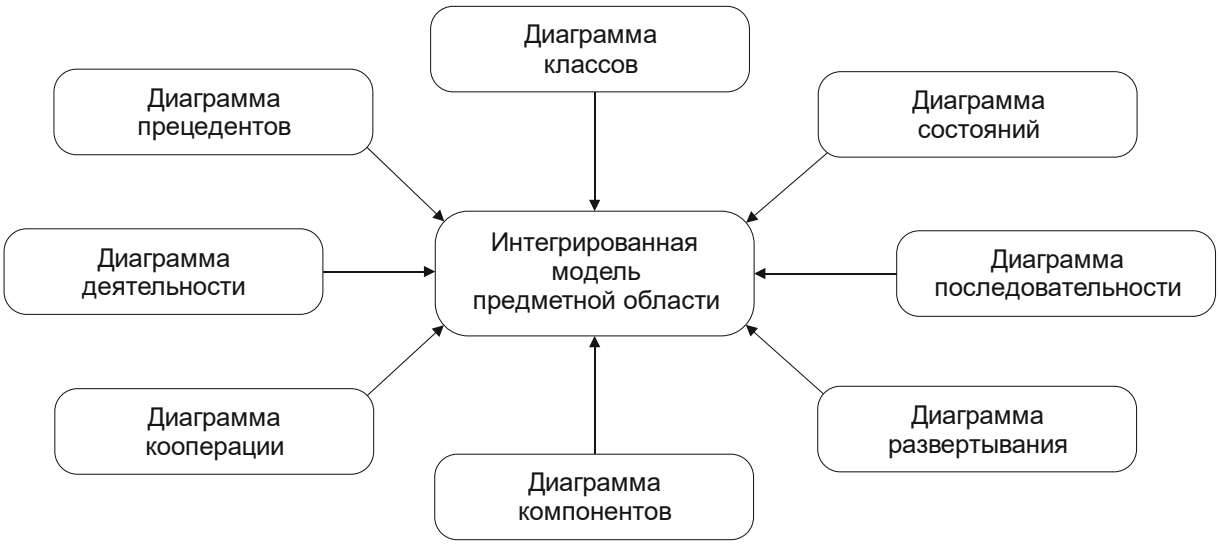


Рис. 2.8 – Интегрированная модель предметной области в нотации UML

Диаграмма прецедентов описывает бизнес в виде графа специального вида, основными элементами которого являются прецеденты, акторы и отношения между ними. *Прецедент* модели бизнеса – это относительно законченная последовательность действий в рамках некоторого бизнес-процесса, приносящая ощутимый результат конкретному действующему лицу (актору). Примеры прецедентов: производство продукта, продажа продукта, сервисное обслуживание, разработка продукта, маркетинг и сбыт. Согласно спецификации UML прецеденты обозначаются эллипсом, внутри которого содержится поясняющий текст, называемый именем прецедента. Имена прецедентов формулируются либо глаголом, либо существительным, обозначающим действие и поясняющим слова.

Акторами в модели бизнеса являются элементы окружения, взаимодействующие с бизнес-процессом и являющиеся его потребителями либо инициаторами. Это может быть физическое лицо – человек, не работающий в компании или работающий в подразделениях, не охваченных моделью бизнеса (клиент, покупатель, поставщик, партнер, акционер, заказчик), юридическое – компания, организация, предприятие, органы власти, являющиеся поставщиками ресурсов либо потребителями продуктов бизнес-процессов. Стандартным обозначением актора является пиктограмма (рис. 2.9), под которой располагается имя актора.

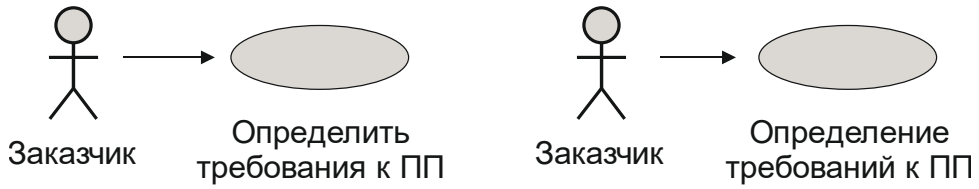
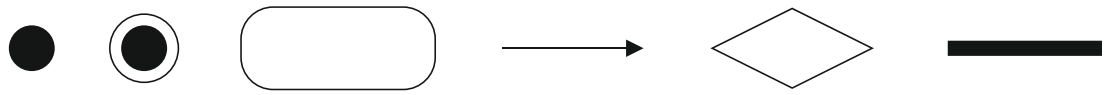


Рис. 2.9 – Эквивалентные прецеденты

Между прецедентами и акторами устанавливаются *отношения коммуникации,* которые описывают информационные, материальные и финансовые потоки между ними. Все прецеденты, вводимые в модель, должны быть связаны с акторами: предметная область не должна содержать бизнес-процессы, которые никем не востребованы.

Наиболее важным для описания прецедента является документ, называемый *потоком событий*. Он описывает сценарии осуществления прецедента в виде последовательности шагов процесса. Поток событий прецедента может быть представлен в виде *диаграммы деятельности*. На рисунке 2.10 приведены условные обозначения основных элементов диаграммы.



а) *б*) *в*) *г*) *д*) *е*)

Рис. 2.10 – Элементы диаграммы деятельности:
а – начальное состояние; *б* – конечное состояние; *в* – действие;
г – переход; *д* – ветвление; *е* – синхронизация

Каждый шаг (событие) прецедента представляет собой некоторое действие, переводящее прецедент в новое состояние. В свою очередь новое состояние прецедента является стимулом для выполнения следующего шага (события).

Пример

На рисунке 2.11 приведен пример диаграммы, иллюстрирующий ход событий прецедента «Продажа продукта».

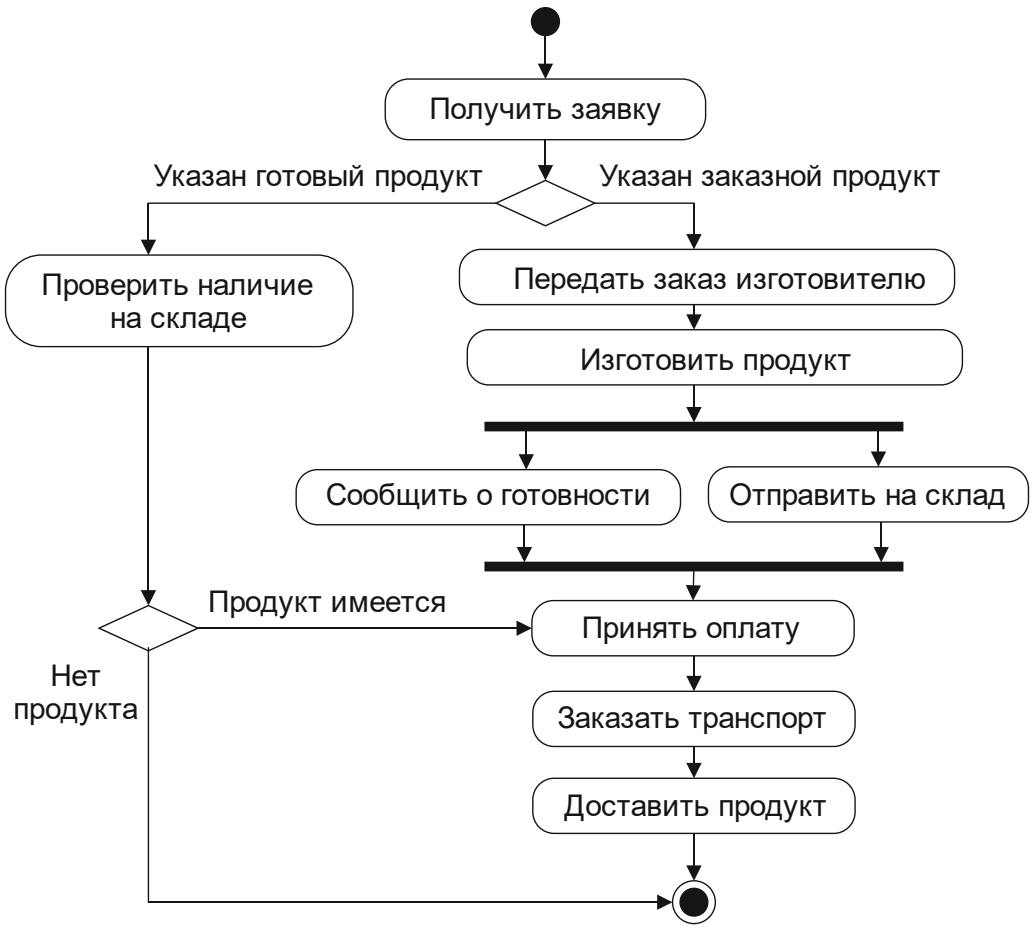


Рис. 2.11 – Диаграмма деятельности прецедента «Продажа продукта»

Объектная модель раскрывает внутреннее устройство бизнеса, а именно: какие виды ресурсов используются для реализации прецедентов и каким образом они взаимодействуют. Основным понятием модели бизнес-процесса является понятие *объекта*. Объекты модели бизнеса представляют акторов (физических лиц), участвующих в выполнении процессов, и различного рода сущности, которые обрабатываются или создаются бизнесом (продукцию, предметы, задачи и т. д.). Участники процессов (исполнители) называются активными объектами, сущности – пассивными.

Пример

Для того чтобы отразить участие акторов во время выполнения бизнес-процессов, используется диаграмма последовательности (рис. 2.12).

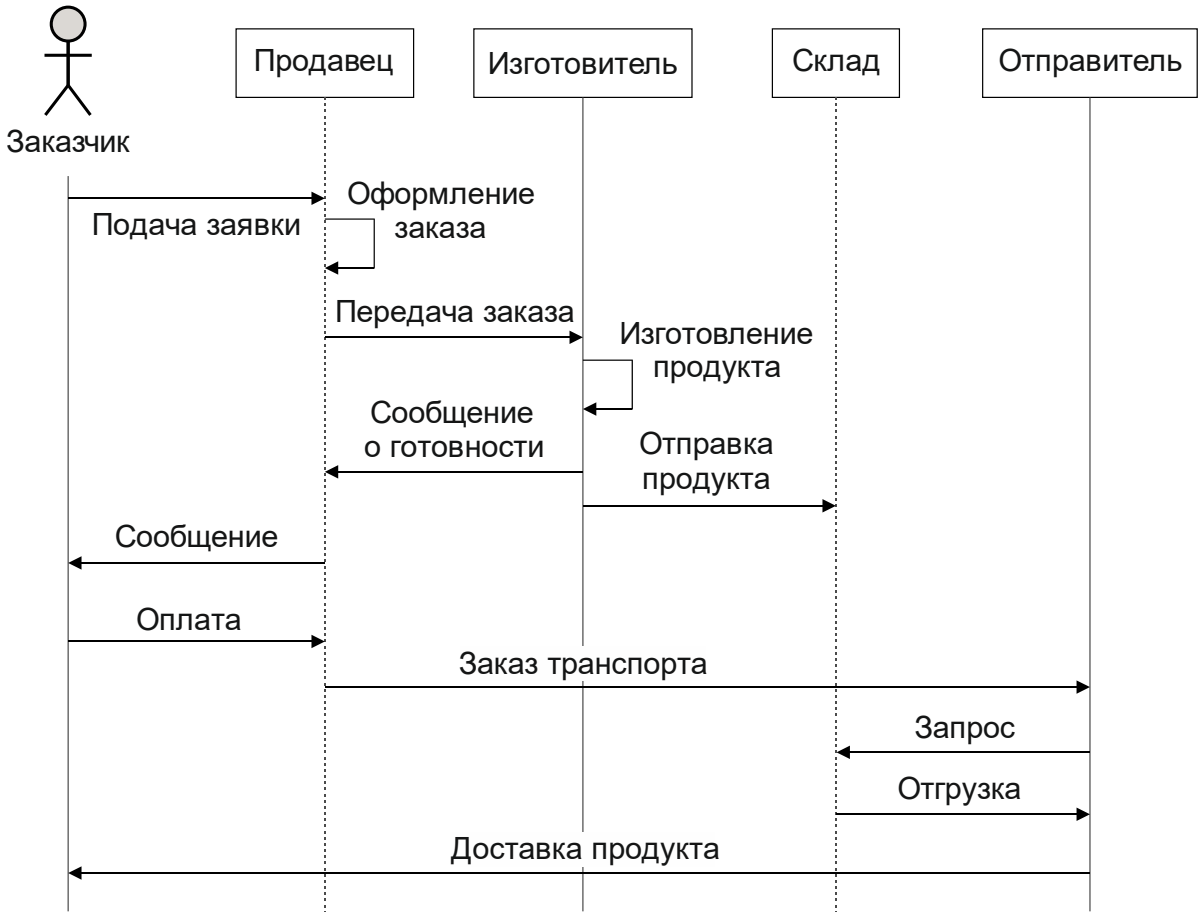


Рис. 2.12 – Диаграмма последовательности прецедента «Продажа продукта»

Каждый актор, участвующий в реализации прецедента, изображается в верхней части диаграммы в виде прямоугольника, от которого вниз проведена

линия («линия жизни»). Внутри прямоугольника записывается имя актора. Между объектами (акторами) устанавливаются отношения сообщений, отражающие аналогично отношениям коммуникации передачу информации (или некоторый материальный поток) между объектами. Сообщение изображается отрезком горизонтальной линии со стрелкой, проведенной от линии жизни объекта (актора), посылающего сообщение, до линии жизни объекта (актора), получающего сообщение. При этом прием сообщения инициирует выполнение определенных действий тем объектом, которому сообщение передано. Сообщения должны быть упорядочены по времени: первое сообщение изображается вверху диаграммы, следующее – ниже, следующее – еще ниже и т. д.

Использование описанных выше методологий описания бизнес-процессов позволяет упростить взаимодействие заказчиков с разработчиками и разработчиков между собой при разработке и анализе требований, проектировании архитектурного и компонентного дизайна ПП и создании в последующем программного кода.

Контрольные вопросы по главе 2

1. Раскройте содержание моделей «Как есть» и «Как должно быть». Что такое ключевой показатель результативности?
2. Раскройте содержание объектно-ориентированного подхода при описании бизнес-процессов предметной области.
3. Раскройте содержание структурного (функционального) подхода при описании бизнес-процессов предметной области.
4. Каковы основные структурные элементы IDEF0-модели?
5. Охарактеризуйте структурные элементы DFD-диаграммы.
6. Что такое прецедент? Что такое актор? Что обозначают эти понятия при моделировании бизнеса?
7. Что такое поток событий прецедента? Как отражается поток событий на диаграмме деятельности языка UML?
8. Что отображается на диаграмме последовательности языка UML?
9. Приведите примеры шаблонов описания требований.
10. Опишите последовательность этапов проектирования архитектуры программной системы управления и контроля работы скорой помощи с использованием диаграмм потоков данных.

3 Модели разработки программного продукта

3.1 Каскадная модель

С точки зрения проектирования и конструирования ПП можно выделить линейные и нелинейные (итеративные) модели разработки ПП. В линейных моделях стадии и процессы разработки ПП следуют строго друг за другом в соответствии с требованиями стандартов на процессы жизненного цикла разработки ПП. Однако в большинстве случаев разработчикам приходится возвращаться на предыдущие стадии чтобы своевременно учесть изменения, возникшие при реализации в проекте. В нелинейных моделях стадии и процессы разработки ПП могут повторяться или совмещаться (происходить одновременно).

К *линейным* моделям относятся каскадная (водопадная) модель и ее разновидность V-образная модель. К *нелинейным* моделям относятся инкрементная и спиральная модели, модель прототипирования, модель быстрой разработки приложений.

В данном разделе приводится описания каскадной модели и модели прототипирования и быстрой разработки приложений [7].

Каскадная модель («*водопад*») является одной из первых, применяемых на практике моделей ЖЦ ПП, в которой каждая работа выполняется один раз в определенной последовательности и с требуемым качеством, после ее завершения и перехода к следующей работе возвращения к предыдущей не требуется (рис. 3.1). Отличительное свойство каскадной модели состоит в том, что она представляет собой формальный метод (разновидность разработки «сверху вниз») и состоит из независимых фаз, выполняемых последовательно.

Каскадную модель можно рассматривать как модель ЖЦ, пригодную для создания первой версии ПП, в следующих случаях: требования к ПП максимально конкретизированы, понятны и не изменяются; разрабатывается новая версия уже существующего продукта, при этом вносимые изменения четко определены; автоматизируются

типовые бизнес-процессы потребителя, содержание которых закреплено нормативными документами.

. . . . . .

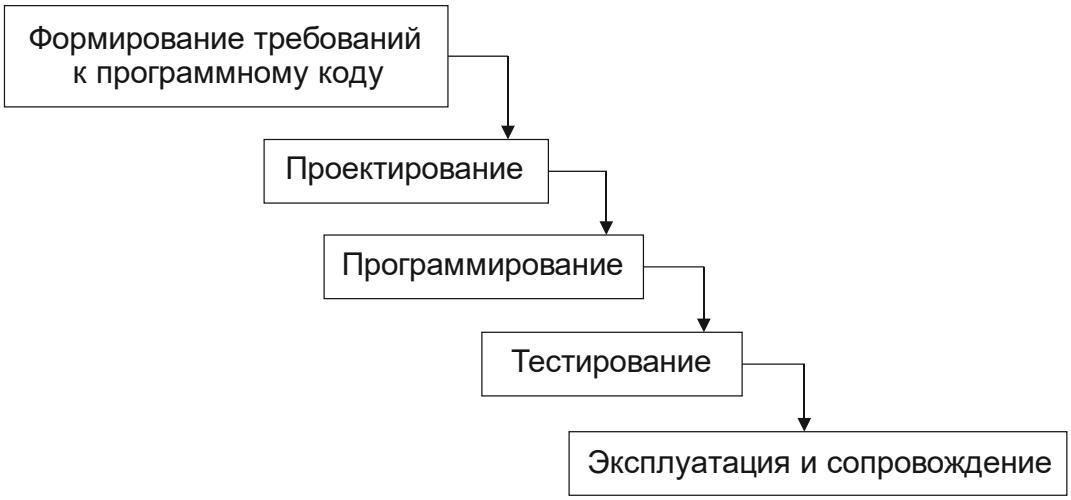


Рис. 3.1 – Жизненный цикл каскадной модели

Основные *положительные моменты* применения каскадной модели заключаются в следующем: модель проста и понятна заказчикам; каждая последующая фаза начинается только после полного завершения предыдущей фазы; на каждой фазе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности; переход от одной фазы к другой осуществляется после приемки-сдачи работ с участием заказчика; выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты. Каскадные модели на протяжении всего времени их существования используются при выполнении крупных проектов, в которых задействовано несколько больших команд разработчиков.

Недостатки каскадной модели особенно остро проявляются в случаях, когда трудно (или невозможно) четко сформулировать требования либо требования меняются в процессе создания продукта. Кроме того, любая попытка вернуться на одну или две фазы назад, чтобы исправить какую-либо ошибку, приводит к значительному увеличению затрат и нарушению сроков разработки; интеграция программных компонентов, в процессе которой обычно выявляется б*о*льшая часть ошибок, выполняется в конце разработки, что сильно увеличивает стоимость их устранения; происходит большое запаздывание с оценкой качества разрабатываемого ПП. В этих случаях разработка ПП имеет циклический характер, когда результаты очередного этапа часто вызывают изменения в проектных

решениях, выработанных на более ранних этапах. Изменения связаны, как правило, с ошибками разработчиков, допущенными на ранних этапах разработки и выявленными на этапе тестирования, а также могут быть вызваны изменениями требований в процессе разработки вследствие неготовности заказчиков правильно их сформулировать либо введением требований, обусловленных изменением бизнес-процессов предметной области. Таким образом, постоянно возникает потребность в возврате к предыдущим фазам и уточнению либо пересмотру ранее принятых решений, в результате чего *реальный процесс разработки* принимает другой вид (рис. 3.2).

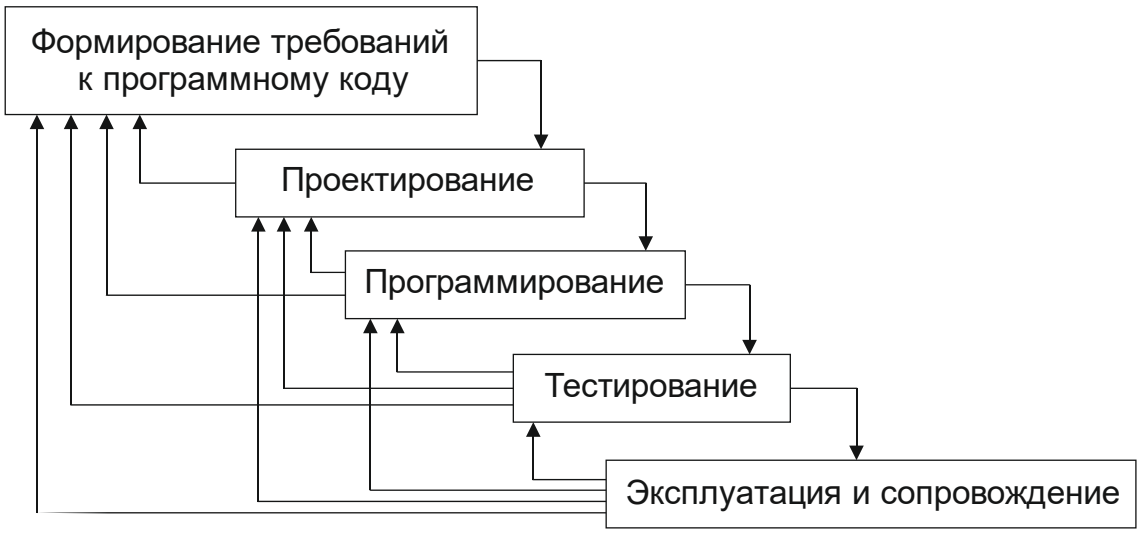


Рис. 3.2 – Модифицированная версия каскадной модели

Модифицированная версия каскадной модели является в значительной степени менее жесткой, чем ее первоначальная форма. В этой версии допускается наличие итераций между фазами (постоянный возврат к предыдущему шагу с целью анализа и проверки на соответствие результатов поставленным задачам), возможность параллельного выполнения фаз и управление изменениями в проекте. Несмотря на то что модифицированная каскадная модель является более гибкой, чем классическая модель, она также не находит широкого применения, так как на практике не удается выполнить приведенные выше условия применения. Попытки оптимизации каскадной модели привели к возникновению нелинейных моделей разработки программного продукта.

3.2 Модель прототипирования

Как правило, под прототипом понимается действующий программный компонент, реализующий отдельные функции и внешние интерфейсы разрабатываемого программного продукта.

Основной идеей модели прототипирования является максимальное вовлечение пользователя в процесс разработки с целью извлечения, описания и корректировки реальных требований к будущему ПП. Это позволяет уже на фазе разработки требований создавать работающий программный компонент, реализующий отдельные функции и внешние интерфейсы разрабатываемого программного продукта. Потенциальные пользователи работают с этим прототипом, определяя его сильные и слабые стороны, о результатах сообщают разработчикам программного продукта. Таким образом, обеспечивается *обратная связь* между пользователями и разработчиками, которая используется для изменения или корректировки спецификации требований к программному продукту. В результате такой работы продукт будет отражать реальные потребности пользователей. Схема ЖЦ модели прототипирования приведена на рисунке 3.3.

Рис. 3.3 – Жизненный цикл модели прототипирования

Начало жизненного цикла разработки помещено в центр эллипса. Жизненный цикл разработки программного продукта начинается с совместной разработки конечными пользователями и разработчиками плана проекта, затем выполняется быстрый анализ предметной области, формулируются предварительные требования, проектируются база данных, пользовательский интерфейс и функционал будущего прототипа программного продукта. В результате этой работы создается *документ*, который содержит частичную спецификацию требований к ПП и в дальнейшем служит основой для итерационного цикла быстрого прототипирования.

Следующий уровень – создание на основе разработанного документа исходного прототипа будущего программного продукта. Далее на каждой итерации прототипирования разработчик демонстрирует пользователям вариант прототипа, а пользователи оценивают его функциональные возможности и определяют проблемы. Этот процесс продолжается до тех пор, пока пользователи не будут удовлетворены степенью соответствия прототипа программного продукта поставленным требованиям. Готовый прототип демонстрируют пользователям, он утверждается и на его основе выполняется разработка программного продукта. Именно на этом этапе ускоренный прототип становится промышленным ПП, удовлетворяющим требованиям заказчика. При разработке производственной версии может понадобиться более высокий уровень реализации функциональных возможностей, подключение различных системных сервисов, необходимых в том числе и для выполнения нефункциональных требований.

После этого следует тестирование в предельных режимах, а затем, как обычно, техническая поддержка процессов функционирования и сопровождения.

Преимущества модели прототипирования состоят в следующем:

- ознакомление заказчика с разрабатываемым ПП начинается на раннем этапе ЖЦ, поэтому снижается вероятность возникновения путаницы, искажения информации или недоразумений при определении требований к программному продукту;
- в процессе разработки всегда можно учесть новые, даже неожиданные требования заказчика, что приводит к созданию более качественного программного продукта;
- прототип представляет собой формальную спецификацию, воплощенную в программный продукт, и позволяет гибко выполнять проектирование и разработку, включая несколько итераций на всех фазах жизненного цикла разработки;

- уменьшается число доработок, что снижает стоимость разработки;
- возникающие проблемы решаются на ранних стадиях ЖЦ, что резко сокращает расходы на их устранение;
- заказчики принимают участие в процессе разработки на протяжении всего жизненного цикла и в конечном итоге несут ответственность за результаты работы наравне с разработчиками.

Основные *недостатки* модели прототипирования:

- прототипирование может продолжаться слишком долго, и разработчики могут попасть в так называемый цикл «кодирование – устранение ошибок», что приводит к дорогостоящим незапланированным итерациям прототипирования;
- разработчики и пользователи не всегда понимают, когда прототип превращается в конечный продукт, поэтому существует необходимость в традиционном документировании процесса;
- на очередной итерации заказчики могут быть удовлетворены качеством прототипа и требуют его немедленной поставки, вместо того чтобы ждать появления полной, хорошо продуманной версии;
- на разработку системы может быть потрачено слишком много времени, так как итерационный процесс демонстрации прототипа и его пересмотр могут продолжаться бесконечно долго, на заказчиков может оказать негативное влияние тот факт, что они не располагают информацией о точном количестве итераций, которые будут необходимы;
- при выборе инструментальных средств прототипирования (операционных систем, технологий проектирования, языков программирования, алгоритмов решения функциональных задач) разработчики могут остановить свой выбор на неэффективных решениях, чтобы продемонстрировать свои способности.

Модель прототипирования рекомендуется применять:

- при выполнении новой, не имеющей аналогов разработки;
- когда заказчик неохотно соглашается на фиксированный набор требований, требования к программному продукту заранее неизвестны и могут уточняться в процессе разработки;

- если разработчики не уверены в выбранных решениях относительно пользовательского интерфейса и функционала, оптимальности применяемой архитектуры или алгоритма.

. . . . . .

3.3 Модель быстрой разработки приложений

В модели быстрой разработки приложений (Rapid Application Development – RAD) пользователь задействован не только при определении требований, но и на всех остальных фазах жизненного цикла разработки ПП: проектировании, кодировании, тестировании, внедрении (рис. 3.4). Для этого необходимо использовать специальное ПО – средства разработки графического пользовательского интерфейса и кодогенераторы. Модель основывается на последовательности итераций создания прототипов, критический анализ которых обсуждается с заказчиком.

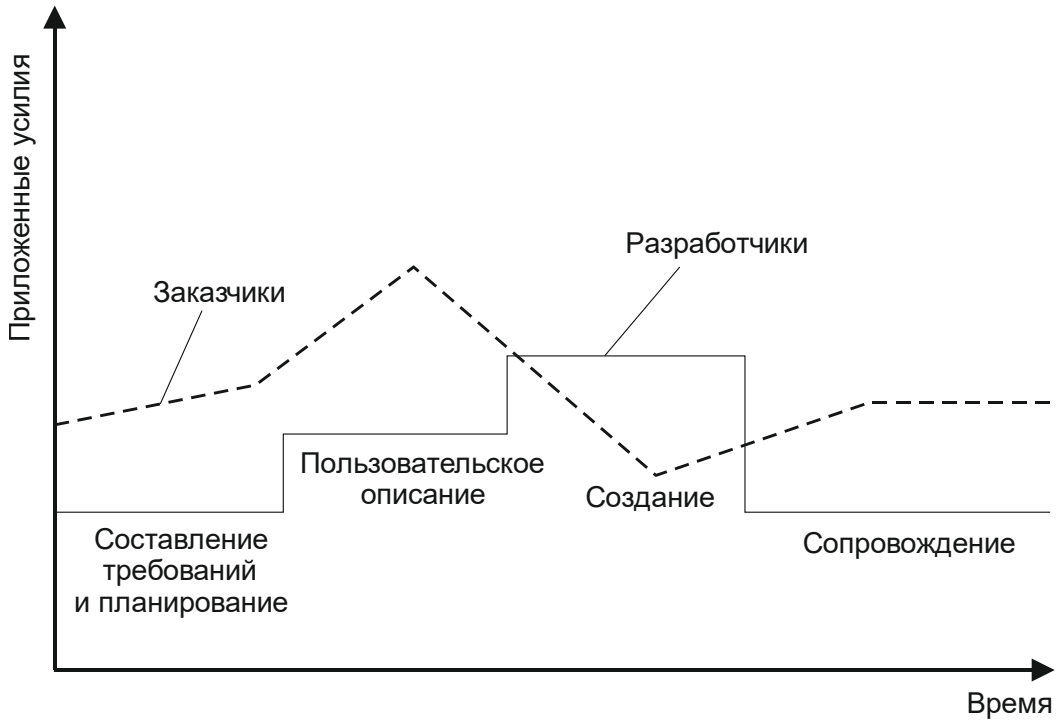


Рис. 3.4 – Модель быстрой разработки приложений

Характерной чертой RAD-модели является короткое время перехода от определения требований до создания полной системы. Разработка каждого интегрированного продукта ограничивается четко определенным периодом времени, который, как правило, составляет 60 дней и называется временным блоком.

В состав каждого временного блока входят анализ, проектирование и внедрение. Факторы, позволяющие создать систему за 60 дней, причем без ущерба

качеству, включают в себя применение мощных инструментальных средств разработки, высокий уровень повторного использования программного кода, быстрый и качественный анализ промежуточных результатов, предоставление необходимых ресурсов.

В RAD-модели конечный пользователь играет *решающую роль*. В тесном взаимодействии с разработчиками он участвует в формировании требований и их апробации на работающих прототипах. Таким образом, в начале жизненного цикла конечный пользователь выполняет большую часть работы, в результате чего создаваемая система формируется быстрее.

RAD-модель включает следующие *фазы*:

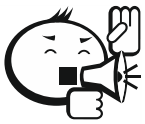
- составление требований и планирование (сбор требований осуществляется с использованием так называемого метода совместного планирования требований (планирование работ по созданию ПП и составление требований к ПП выполняются одновременно), который заключается в структурном анализе и обсуждении решаемых задач (будущего функционала);
- пользовательское описание (проектирование ПП, выполняемое при непосредственном участии заказчика, при этом работающая над проектом команда зачастую использует специальные инструментальные средства, обеспечивающие сбор пользовательской информации);
- создание (детальное проектирование, кодирование и тестирование ПП и его поставка заказчику за определенное время);
- сопровождение (проведение пользователем приемочных испытаний, установка ПП и обучение пользователей).

Достоинства RAD-модели состоят в следующем:

- использование современных инструментальных средств позволяет сократить время ЖЦ разработки;
- постоянное присутствие заказчика сводит до минимума риск неудовлетворения продуктом и гарантирует соответствие системы коммерческим потребностям и надежность программного продукта в эксплуатации;
- основное внимание переносится с разработки документации на создание кода;
- повторно используются компоненты уже существующих программ.

В то же время RAD-модели присущи и *недостатки*:

- если заказчики не могут постоянно участвовать в процессе разработки, то это может негативно сказаться на качестве программного продукта;
- для работы нужны высококвалифицированные кадры как разработчиков, так и пользователей, умеющих работать с современными инструментальными средствами;
- использование модели может оказаться неудачным при отсутствии пригодных для повторного использования компонентов;
- для реализации модели требуются разработчики и заказчики, которые готовы к быстрому выполнению действий ввиду жестких временных ограничений;
- команды, разрабатывающие коммерческие проекты с помощью модели RAD, могут «затянуть» разработку программного продукта до такой степени, что его поставка конечному пользователю будет под большим вопросом;
- существует риск, что работа над проектом никогда не будет завершена, в связи с этим менеджер проекта должен сотрудничать как с командой разработчиков, так и с заказчиком, что позволит избежать появления замкнутого цикла.



RAD-модель можно применять при разработке программных продуктов, хорошо поддающихся моделированию, когда требования к ПП хорошо известны, а заказчик может непосредственно участвовать в процессах разработки ПП на всех этапах ЖЦ.

Контрольные вопросы по главе 3

1. Раскройте достоинства и недостатки каскадной модели разработки ПП.
2. Поясните, в каких случаях целесообразно использовать каскадную модель разработки ПП.
3. Раскройте достоинства и недостатки модели прототипирования.
4. Поясните, в каких случаях целесообразно использовать модель прототипирования.

5. Раскройте достоинства и недостатки модели быстрой разработки приложений.
6. Поясните, в каких случаях целесообразно использовать модель быстрой разработки приложений.

4 Жизненный цикл разработки программного продукта

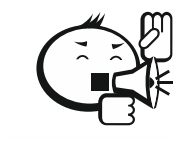
4.1 Стандарты на процессы жизненного цикла разработки программного продукта

Основными стандартами, регламентирующими последовательность и содержание процессов жизненного цикла разработки программного продукта, являются:

- Единая система программной документации (ЕСПД): ГОСТ 19.102–77 ЕСПД «Стадии разработки»;
- IEEE-1074–1997 «Процессы и действия жизненного цикла программного обеспечения» (Developing software life cycle processes);
- ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств».

Стандарт «Единая система программной документации» (ЕСПД) представляет собой комплект из 23 документов, составляющих систему межгосударственных стандартов стран СНГ (ГОСТ 19), действующих на территории Российской Федерации, на основе межгосударственного соглашения по стандартизации. Несмотря на то что большая часть комплекса ЕСПД была разработана в 1970–1980-е гг., стандарт пользуется большой популярностью как у разработчиков ПП, так и у организаций, планирующих участие в конкурсе (тендере) на разработку ПП.

В стандарте все действия по разработке программных продуктов и программной документации независимо от их назначения и области применения подразделяются на стадии, этапы и работы. В качестве основных этапов в стандарте выделены: разработка технического задания, разработка эскизного проекта, разработка технического проекта, разработка рабочего проекта, приемка-сдача ПП заказчику.



Основным достоинством стандарта является то, что в нем четко и компактно прописаны виды программных документов и их краткое содержание, которые должны быть представлены заказчиками по окончании контракта: техническое задание, исходный текст программы, описание программы, ведомость эксплуатационных документов, руководство системного программиста, руководство программиста, руководство пользователя, руководство по технической поддержке пользователя. Обязательным документом, определенным стандартом, является хорошо задокументированный исходный текст программы. Все остальные документы оговариваются в контракте на разработку (поставку).

В стандарте IEEE[1] 1074–1997 «Процессы и действия жизненного цикла программного обеспечения» (Developing a software project life cycle processes) содержание ЖЦ разработки ПП описывается набором из 6 фаз, 17 процессов и 65 действий.

Фаза определяется как группа логически связанных процессов, в ходе осуществления которых выполняется вполне конкретная часть проекта. *Процесс* представляет собой ряд действий (работ), выполнение которых приводит к конкретному результату. *Действие* (работа) – деятельность, выполняемая в процессе реализации проекта. При этом каждое действие характеризуется продолжительностью, стоимостью и потребностями в ресурсах и может быть детализировано командой разработчиков на более мелкие составляющие (задачи). Содержание стандарта не отождествляется с конкретной моделью ЖЦ разработки ПП и не предполагает использование определенной методологии разработки. Последовательность фаз, рекомендуемых стандартом: выбор модели ЖЦ разработки ПП, управление проектом, предварительная разработка проекта (концепции проекта), разработка проекта, сопровождение проекта, интеграция проекта.

Основу стандарта ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств» составляют следующие базовые понятия:

- *стадия* – период в пределах ЖЦ ПП, который относится к описанию или реализации одного из конкретных состояний;

[1]Читается «ай-трипл-и».

- *процесс* – совокупность взаимосвязанных или взаимодействующих видов деятельности, преобразующих исходные данные о ПП либо его отдельных компонентах в выходные результаты;
- *деятельность* – совокупность действий, используемых для получения конкретных выходных результатов;
- *задача* – элементарное действие, предназначенное для достижения одного или более выходных результатов процесса, при выполнении которого можно однозначно назначить исполнителя и определить требуемые ресурсы. Задача формулируется в форме требования, рекомендации или допустимого действия, при этом используются глаголы: «*должен*» – для выражения условия, требующего проверки на соответствие чему-либо; «*следует*» – выражение рекомендации среди других возможностей; «*может*» – чтобы отразить направление допустимых действий.

Модель жизненного цикла ПП – структура, состоящая из процессов, действий и задач, включающих разработку, эксплуатацию и сопровождение программного продукта, охватывающая период существования ПП – от установления требований к ПП до полного прекращения его использования.

Стандарт содержит полный набор описания процессов ЖЦ ПП для некоторого типового проекта с максимально возможным составом процессов, действий и задач, которые используются: при приобретении системы, содержащей программные средства, или отдельно поставляемого ПП; при оказании программной услуги; при поставке, разработке, эксплуатации и сопровождении ПП. Последовательность действий или задач в каждом процессе не является жесткой и определяется как логическими связями между ними, так и используемой при создании ПП модели разработки.

Пример

На рисунке 4.1 представлен один из возможных вариантов ЖЦ создания ПП, характерного для каскадной модели разработки.

Рис. 4.1 – Вариант ЖЦ создания ПП каскадной модели разработки

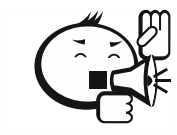
4.2 Разработка и анализ требований

4.2.1 Понятие и классификация требований

В общем случае требования должны описывать потребности людей, заинтересованных в создании ПП, и описывать условия или возможности системы в целом либо ее отдельных компонентов, необходимые пользователям для решения своих проблем и исполнения должностных обязанностей.

Как правило, требования оформляются в виде специального документа – технического задания.

Существует несколько «моделей» классификации требований, представленных в различных документах.



В [8] множество требований условно разбиты на следующие группы:

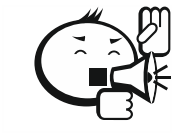
Требования к продукту и процессу должны описывать свойства продукта, который необходимо получить, и процесса, с помощью которого продукт будет создаваться.

Функциональные требования характеризуют функциональные возможности программного обеспечения, методы передачи и преобразования входных данных в результаты.

Нефункциональные требования определяют условия и среду выполнения функциональных требований, например, защита и доступ к БД, взаимодействие компонентов и др.

Системные требования описывают требования к программному продукту, состоящему из взаимосвязанных программных и аппаратных подсистем и разных приложений. Требования могут оцениваться количественно (например, количество запросов в единицу времени), значительная часть требований относится к атрибутам качества: безотказность, надежность и др.

В серии стандартов *ГОСТ 34.602–89* «*Информационная технология. Техническое задание на создание автоматизированных систем*» не приводятся типы требований, однако определен состав и правила оформления документа «Техническое задание на создание системы». Данный документ устанавливается как основной документ, определяющий «требования и порядок создания автоматизированной системы», предполагая, что на основании этого документа будет производиться разработка и приемка системы.



Классифицировать требования в рамках ГОСТ 34.602 можно на основе определяемой им структуры технического задания:

1. Требования к функциям (задачам), выполняемым системой.
2. Требования к структуре и режимам функционирования системы.
3. Требования к видам обеспечения: математическое обеспечение; информационное обеспечение; программное обеспечение; техническое обеспечение; организационное обеспечение.
4. Общие требования к приемке работ по стадиям, порядок согласования и утверждения приемочной документации.

5. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие.
6. Требования к документированию системы.

Классификация, предложная К. Вигерсом, содержит следующие уровни требований (рис. 4.2) [9].

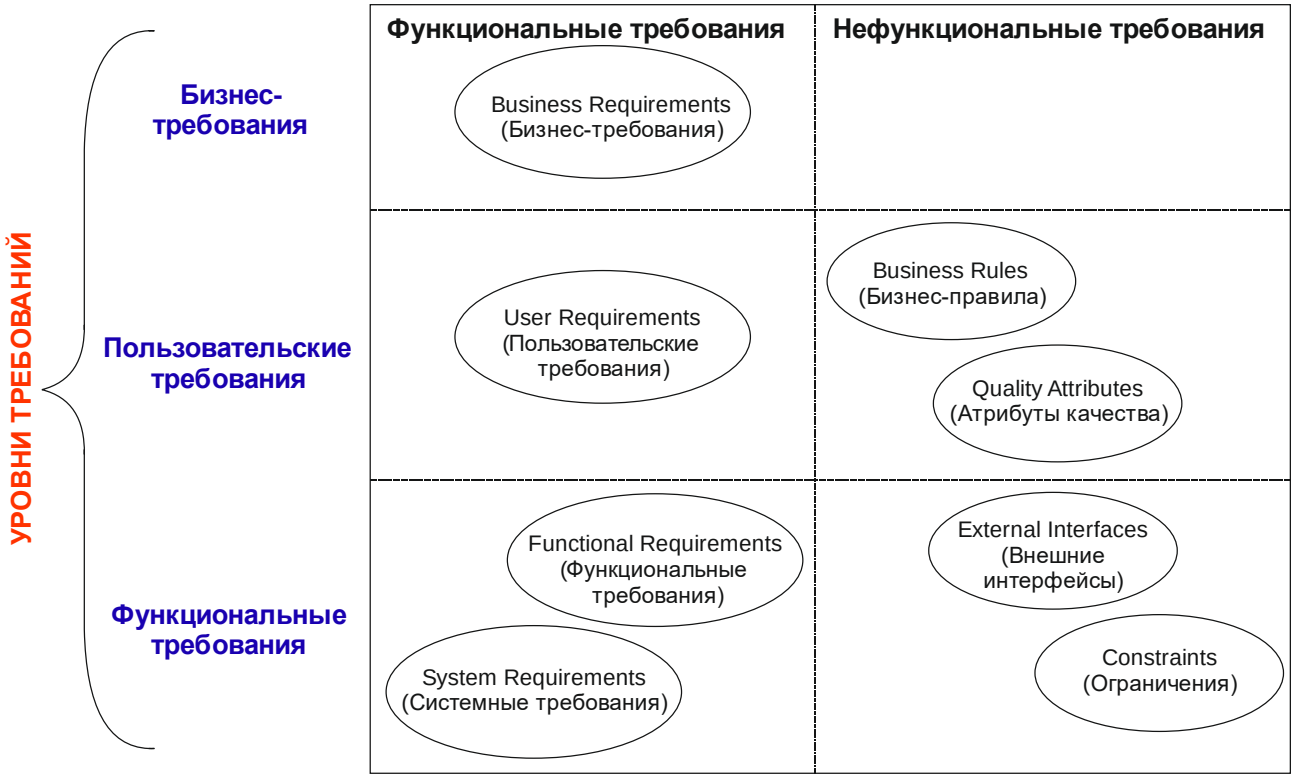


Рис. 4.2 – Классификация Вигерса

Состав и содержание бизнес-требований должны определяться исходя из выявленных проблем в управлении бизнес-процессами у организации-заказчика и описания цели организации – заказчиков системы. Как правило, это происходит во время встреч руководителей организации с системными аналитиками и архитекторами проектов. Требования на данном уровне объясняют, зачем организации нужна такая система, какие цели бизнеса организация намерена достичь при внедрении ПП.

Таким образом, к бизнес-требованиям относится вся информация, описывающая финансовые, рыночные или другие отношения коммерческого характера, которые потенциальные заказчики собираются получить от использования продукта.

Ниже приводятся примеры формулировок проблем и соответствующих им бизнес-требований.

Пример

Проблемы:

- высокий уровень запасов сырья, материалов и комплектующих на складе, что приводит к большим объемам оборотных средств предприятия;
- низкий уровень качества планирования учета и контроля движения сырья, материалов и комплектующих на складе.

Бизнес-требования:

- сократить уровень показателя «объем оборотных средств предприятия» на 10%;
- повысить уровень качества планирования учета и контроля движения сырья, материалов и комплектующих на складе за счет внедрения математических моделей управления запасами.

Пользовательские требования должны быть ориентированы на выполнение бизнес-требований и описывать задачи (*возможности*)*, которые система позволит решить пользователям в рамках исполнения своих служебных обязанностей* (*должностных инструкций*).

Участники выявления требований: сотрудники компании (потенциальные пользователи ИТ, архитектор ИТ, системный аналитик).

Пример

Варианты формулировок пользовательских требований выглядят следующим образом:

- «персонал службы производственного отдела должен иметь возможность решать задачу планирования размеров производственных запа-

сов исходного сырья, материалов и комплектующих с учетом ограничений на объемы оборотных средств и обеспечения непрерывности и ритмичности производства готовой продукции»;

- «персонал службы мониторинга должен иметь возможность решать задачу мониторинга, учета и контроля движения сырья, материалов и комплектующих на складе».

Структуризация требований пользователей осуществляется при помощи функциональных требований к системе, которые определяют функциональные возможности программного обеспечения, методы передачи и преобразования входных данных в результаты, которые разработчики должны реализовать, чтобы выполнить бизнес-требования и требования пользователей.

Разработка функциональных требований к программному комплексу осуществляется системным архитектором на основе анализа и детализации требований пользователей.

Пример

Примеры *функциональных требований*: «программный комплекс ПК1 должен обеспечить сбор, обработку, хранение и защиту информации при решении задачи планирования размеров производственных запасов исходного сырья, материалов и комплектующих с учетом ограничений на объемы оборотных средств и обеспечения непрерывности и ритмичности производства готовой продукции».

Нефункциональные требования определяют атрибуты качества программного продукта и его отдельных компонентов: надежность, практичность, эффективность, сопровождаемость, мобильность.

Классификация требований позволяет снизить сложность процесса анализа, структурировать работу в рамках проекта, помогает правильно назначить

приоритеты при реализации требований, использовать общие шаблоны для работы с требованиями одного типа. Кроме того, группируя требования по уровням и типам, можно эффективно использовать инструментальные средства, предназначенные для работы с требованиями, в том числе для автоматической генерации документации требований. В свою очередь при «бесструктурном» представлении требований сложно проводить анализ влияния при изменении требований ввиду проблематичности установления связей трассировки. Отсутствие структуры при работе с требованиями порядком усложнит тестирование созданного ПО, ввиду сложности трассировки требований со сценариями тестирования.

4.2.2 Процессы работы с требованиями

Процессы работы с требованиями инициируются в начале проекта и продолжаются на протяжении всего жизненного цикла, вплоть до его завершения, и заключаются в преобразовании выявленных совместно с заказчиком требований в описание требований к программному продукту, его спецификацию и верификацию. На данной стадии необходимо осуществлять следующие виды деятельности: извлечение требований, анализ требований, спецификация требований, аттестация требований, управление требованиями [9].

Раздел «*Извлечение* (*выявление*) *требований*» освещает вопросы сбора требований с точки зрения как организации процесса, так и определения источников, откуда поступают требования. Определение заинтересованных лиц в создании ПО, выявление их служебных обязанностей, описание бизнес-процессов – все это является ключевыми вопросами, без четкого и однозначного ответа на которые даже не стоит думать об успешности проекта.

Один из ключевых принципов программной инженерии заключается в обеспечении взаимодействия между пользователями и разработчиками. Прежде всего, специалисты «по требованиям» – системные аналитики определяют способы коммуникаций и взаимопонимания между заказчиками и исполнителями, которые необходимы для решения задач проекта. Далее необходимо идентифицировать все возможные источники требований, значимые для решения задач проекта (описание функциональных обязанностей, должностных инструкций, договоров, материалов аналитиков по задачам и функциям системы и т. д.). Идентифицировав источники требований, необходимо перейти к сбору требований с использованием методов: дерева целей, анкетирования, разработки сценариев, интервьюирования и т. д.

Раздел «*Анализ требований*» посвящен описанию процессов анализа требований, то есть трансформации информации, полученной от пользователей (и других заинтересованных лиц), в четко и однозначно определенные требования, передаваемые разработчикам для реализации в программном коде.

Анализ требований включает:

1) процессы изучения потребностей и целей пользователей;
2) классификацию требований и их преобразование к требованиям программного обеспечения и общесистемному аппаратно-программному обеспечению;
3) установление и разрешение конфликтов между требованиями;
4) определение приоритетов требований с точки зрения их внешней согласованности со средой функционирования, внутренней согласованности между программными элементами, тестируемости, реализуемости в составе программного проекта, влияния на процессы функционирования и сопровождения ПП.

Результаты этапа отражаются в специальном документе – техническом задании. Именно этот документ будет определять технические спецификации программного продукта, его функциональность и требования к эксплуатационным характеристикам. Любая ошибка или любое упущение, допущенные при разработке ТЗ, приводят к дополнительным затратам на исправление или доработку готового программного продукта.

Спецификация требований заключается в определении структуры ПО, требований к функциям, качеству и документации, описании в общих чертах архитектуры ПО, алгоритмов решения задач и обработки информации, логики управления и структуры данных, требований к взаимодействию с другими компонентами и платформами.

Проверка (*аттестация*) *требований.* Аттестация требований – это процесс проверки правильности спецификаций требований, их непротиворечивости, полноты и выполнимости, а также соответствия стандартам. На этом этапе заказчик и разработчик ПО проводят экспертизу сформированного варианта требований, с тем чтобы разработчик мог далее проводить разработки. В результате проверки требований подписывается согласованный выходной документ, устанавливающий полноту и корректность требований, а также возможность продолжить процессы проектирования. Одним из методов аттестации является прототипирование, т. е. быстрая реализация отдельных требований в виде прототипа

будущего ПО, анализ масштабов изменения требований, измерение объема функциональности, трудозатрат и стоимости.

Управление требованиями. Требования часто меняются в силу изменения внешних условий и внутренних бизнес-процессов. Необходимо понимать неизбежность изменений и планировать шаги по уменьшению проблем, связанных с изменениями. Данный процесс, точнее комплекс процессов, охватывает весь жизненный цикл программного обеспечения. Управление изменениями, сопровождение и поддержка актуальности требований и их реализации – ключ к успешным процессам программной инженерии. Управление изменениями не может быть хаотическим, поэтому к управлению требованиями нужно относиться как к постоянно действующему бизнес-процессу. Восприятие изменений и возможность их своевременной обработки – вопрос способности проектной команды работать в постоянно меняющихся условиях. Так или иначе, понимание меняющейся природы требований – один из факторов адекватного реагирования на сами изменения, а следовательно, и возможности успешного завершения проекта.

На данной стадии необходимо осуществлять следующие виды деятельности: определение требований к программным компонентам и программному продукту и их интерфейсам; анализ требований к программным компонентам и программному продукту на корректность и тестируемость; определение влияния требований к программным компонентам и программному продукту на среду функционирования; установление совместимости и взаимосвязи между требованиями к программным компонентам и требованиями к программному продукту; определение приоритетов реализации требований к программному продукту и его компонентам; оценку изменения в требованиях к программному продукту и его компонентам по стоимости, времени выполнения работ и воздействиям на технические характеристики, доведение до сведения заинтересованных сторон требований к программным компонентам и программному продукту.

На основании требований к ПП разрабатывается подробное *техническое задание* (ТЗ). Именно этот документ будет определять технические спецификации программного продукта, его функциональность и требования к эксплуатационным характеристикам. Любая ошибка или любое упущение, допущенные при разработке ТЗ, приводят к дополнительным затратам на исправление или доработку готового программного продукта.

4.3 Проектирование программных продуктов

4.3.1 Содержание этапа проектирования

Основная цель процесса проектирования – преобразование общих внешних требований к системе и описание предметной области в конкретные модели программного продукта. Как правило, модель предметной области и требования могут быть транслированы в артефакты системы (*в том числе и программный код*) *множеством способов. Задача архитектора программного обеспечения заключается в выборе и реализации наиболее подходящего способа осуществления необходимых действий по преобразованию.*

Процесс проектирования принято разделять на два этапа: *предварительное проектирование* (архитектурное проектирование); *детальное проектирование* [10].

На *предварительном этапе проектирования* определяется общая архитектура (архитектурный дизайн) программной системы, состав программной системы в виде программных компонент и модулей, интерфейсы между компонентами и способы взаимодействия компонент.

При разработке архитектуры все требования к программному продукту распределяются по программным компонентам и в дальнейшем уточняются для облегчения детального проектирования. На данном этапе принимаются решения, которые являются наиболее важными, определяющими для системы.

Принято считать, что предварительное проектирование включает три типа деятельности:

Структурирование системы – система структурируется на несколько подсистем, где под подсистемой понимается независимый программный компонент, определяются взаимодействия подсистем.

Моделирование управления – определяется модель связей управления между частями системы.

Декомпозиция подсистем на модули – каждая подсистема разбивается на модули, определяются типы модулей и межмодульные соединения.

В процессе *детального проектирования* выполняется декомпозиция архитектурного дизайна до элементарных программных компонентов, определяются состав и структура каждого компонента и его описание в объеме, необходимом для верификации относительно установленных требований к архитектуре программного продукта. Решения, принимаемые на данном этапе, носят локальный характер и могут быть переработаны во время реализации.

Стоимость ошибок архитектурного проектирования очень высока. Детальное проектирование, напротив, нередко осуществляют сами программисты в процессе разработки, если их квалификация это позволяет. Например, алгоритм конкретной процедуры или дизайн конкретной пользовательской формы – типичные рабочие проектные решения, выполняемые программистами в процессе программной реализации. Детальное проектирование зачастую сливается с процессом кодирования.

Для успешной реализации проекта объект проектирования (программная система) должен быть, прежде всего, адекватно описан, т. е. должны быть построены полные и непротиворечивые модели архитектуры ПС, обусловливающей совокупность структурных элементов системы и связей между ними, поведение элементов системы в процессе их взаимодействия, а также иерархию подсистем, объединяющих структурные элементы.

Под моделью понимается полное описание системы ПО с применением структурной или объектно-ориентированной методологии проектирования, представляющих собой средства для визуализации, описания, проектирования и документирования системы.

В обоих случаях при описании структуры и поведения программной системы и взаимосвязей между ее элементами используются два типа нотаций: структурные и поведенческие.

Структурные нотации являются графическими, они используются для представления структурных аспектов проектирования, компонентов и их взаимосвязей, элементов архитектуры и их интерфейсов. Нотации включают языки описания архитектуры и интерфейса, диаграммы классов и объектов, диаграммы

«сущность – связь», компонентов, развертывания, а также структурные диаграммы и схемы. Такие нотации создаются с использованием формальных языков проектирования.

Поведенческие нотации отражают динамический аспект поведения систем и их компонентов. Таким нотациям соответствуют диаграммы:

- Data Flow – один из основных инструментов структурного анализа и проектирования;
- Decision Tables – способ компактного представления модели со сложной логикой;
- Activity – диаграмма поведения, на которой показан автомат и подчеркнуты переходы потока управления от одной деятельности к другой;
- Colloboration – диаграмма поведения, на которой показано взаимодействие и подчеркнута структурная организация объектов, посылающих и принимающих сообщения;
- Sequence – диаграмма поведения, на которой показано взаимодействие и подчеркнута временная последовательность событий и др.

Результаты моделирования позволяют понять и осмыслить структуру и поведение будущей системы, облегчить управление процессом ее создания и уменьшить возможный риск, а также документировать принимаемые проектные решения.

Существуют два принципиально различающихся подхода к моделированию в процессе проектирования систем. Первый подход (*нисходящее проектирование*) основан на анализе предметной области с постепенным выявлением требуемых функций проектируемой системы. Второй подход (*восходящее проектирование*) основан на постепенном синтезе проектируемой системы из обрывочных требований к ней.

При нисходящем проектировании декомпозиция исследуемого объекта (предметной области, проектируемой программной системы) продолжается до тех пор, когда бизнес-логика отдельных компонентов будет ясна и можно приступать к этапу конструирования. При проектировании по направлению от общих объектов к конкретным путем декомпозиции, шаг за шагом уточняются знания об объекте автоматизации и требуемой программной системе, при этом на начальном этапе не приходится иметь дело со слишком большим количеством информации. Такой процесс итеративен: после выполнения одного этапа декомпозиции разработчики переходят к следующему уровню декомпозиции, и таких

уровней обычно несколько. При этом использование различных моделей декомпозиции позволяет получать и сравнивать различные варианты структуры, выявляя их общие закономерности.

При использовании восходящего подхода в первую очередь идентифицируются мелкие очевидные компоненты, из которых должна состоять программная система, и определяются их функции. Затем между компонентами находятся общности, определяются интерфейсы и выполняется их синтез (композиция), получаются более крупные, абстрактные объекты и т. д. Использование восходящего проектирования целесообразно в тех случаях, когда существует необходимость создать компактную систему, функционирование которой строго ограничено требованиями к ней.

В литературе отмечается, что нельзя рассматривать восходящее и нисходящее проектирование как две конкурирующие стратегии, они скорее дополняют друг друга. Более того, сложно на практике рассматривать применимость восходящего проектирования как самостоятельной стратегии. Практически всегда, получив проект ПС целой системы, разработчики с целью проверки принятых проектных решений вынуждены спуститься вниз путем ее декомпозиции.

На этапе предварительного проектирования, как при нисходящем, так и при восходящем проектировании используется *концепция слоев*, в основу которой положено понятие *модульности* (свойство системы подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга модулей). *Модуль* – это подпрограмма (функция или процедура), оформленная определенным образом и выполняющая строго одно действие, это фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации. Разбиение системы на модули позволяет разработчику в некоторый момент времени концертировать свое внимание на небольших, обособленных фрагментах кода, не вникая в подробности реализации остальных фрагментов.

4.3.2 Типовые архитектуры программных систем

Физическая архитектура программных систем (ее программно-аппаратная часть) может быть представлена в виде следующих структур: архитектура «файл-сервер»; двухзвенная архитектура «клиент-сервер»; многозвенная архитектура «клиент-сервер»; архитектура

веб-приложений; архитектура распределенных систем; сервис-ориентированная архитектура.

. . . . . .

Следует заметить, что, как и любая классификация, данная классификация архитектур не является абсолютно жесткой. В архитектуре любой конкретной программной системы можно найти влияние нескольких общих архитектурных решений.

Архитектура файл-сервер предполагает наличие трех основных компонент: файлового сервера, файлового клиента и локальной сети для общения между ними. Файловый сервер – это комплекс аппаратных и программных средств, обеспечивающий совместный доступ к файловым ресурсам (а также к принтерам) через локальную сеть многим пользователям одновременно. Файловый клиент – это набор программного обеспечения, обеспечивающий доступ к файловым ресурсам сервера (или серверов) с персонального компьютера. Клиент устанавливается на каждом рабочем месте, с которого должен осуществляться доступ к серверу. Работоспособность файл-серверного приложения напрямую зависит от надежности и производительности локальной сети.

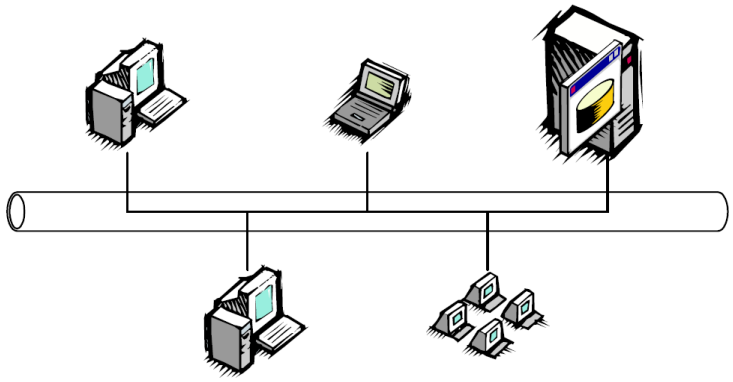


Рис. 4.3 – Архитектура клиент-сервер

Согласно словарю компьютерных терминов, *клиент-сервер* – это вид распределенной системы, в которой есть сервер, выполняющий запросы клиента, причем сервер и клиент общаются между собой с использованием того или иного протокола. Как и для файл-серверной архитектуры, составными компонентами клиент-серверной архитектуры являются сервер, клиентские места и сетевая инфраструктура. Однако, в отличие от предыдущего случая, сервер здесь является уже не сервером файлов, а сервером баз данных или даже сервером приложений. Таким образом, на сервер ложится не просто задача хранение файлов, а поддержание базы данных в целостном состоянии или, в случае сервера приложений,

даже выполнение той или иной части прикладной задачи. Соответственно, общение между клиентом и сервером происходит не на уровне файлов, а на уровне обмена запросами. Клиент передает серверу высокоуровневые запросы на получение той или иной информации либо на ее изменение, а сервер возвращает клиенту результаты выполнения запросов. С точки зрения количества составных частей клиент-серверные системы делятся на *двухуровневые* и *трехуровневые*.

Двухуровневые системы состоят только из клиента и сервера. В трехуровневых же между пользовательским клиентом и сервером, осуществляющим хранение и обработку базы данных, появляется третий, промежуточный слой, являющийся для пользователя сервером, а для системы управления базами данных – клиентом (рис. 4.4). Такая архитектура позволяет более гибко распределять функции системы и нагрузку между компонентами программно-аппаратного комплекса, а также может снизить требования к ресурсам рабочих мест пользователей.

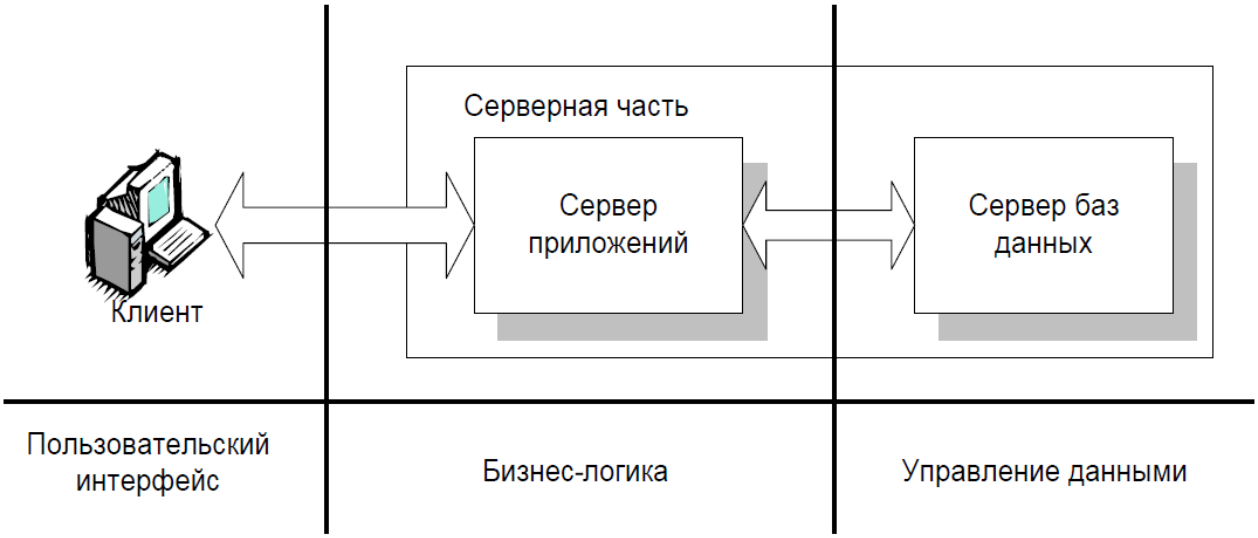


Рис. 4.4 – Трехуровневый «клиент-сервер»

Развитием архитектуры клиент-сервер является так называемая многоуровневая архитектура (Multitier architecture). В данном случае выделяют много серверов приложений (и возможно серверов баз данных), взаимодействующих друг с другом. При этом каждый сервер решает свою бизнес-задачу (рис. 4.5).

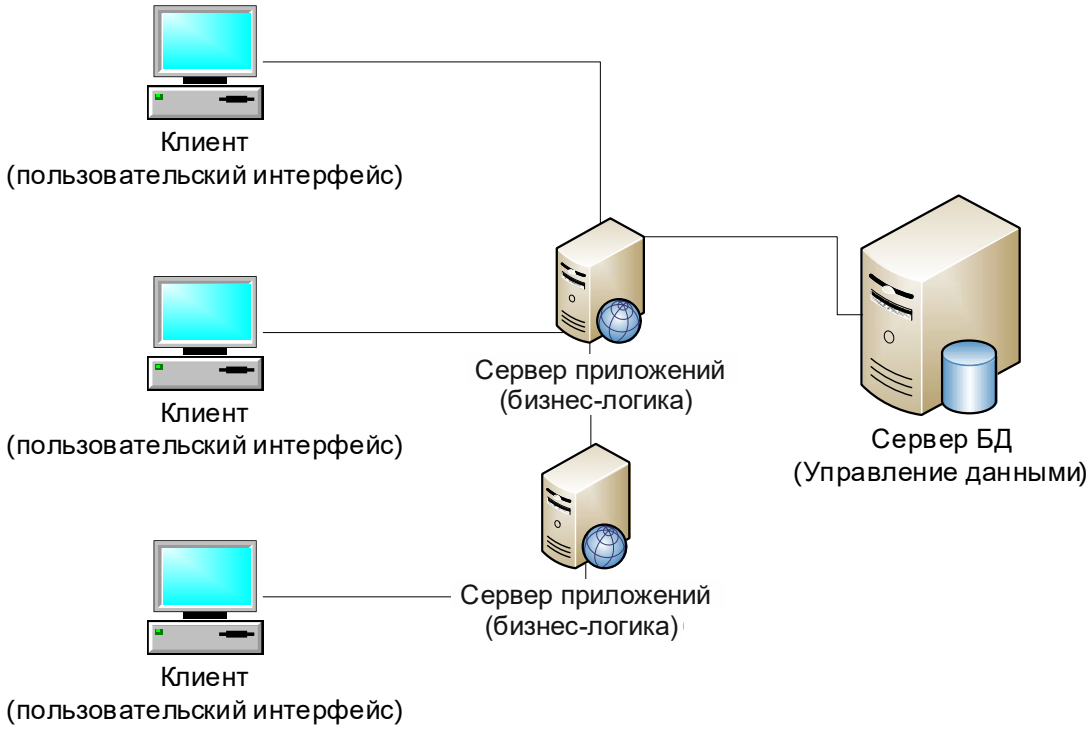


Рис. 4.5 – Многоуровневая клиент-серверная архитектура

Архитектура *веб-приложений* (рис. 4.6) является частным случаем многоуровневой клиент-серверной архитектуры.

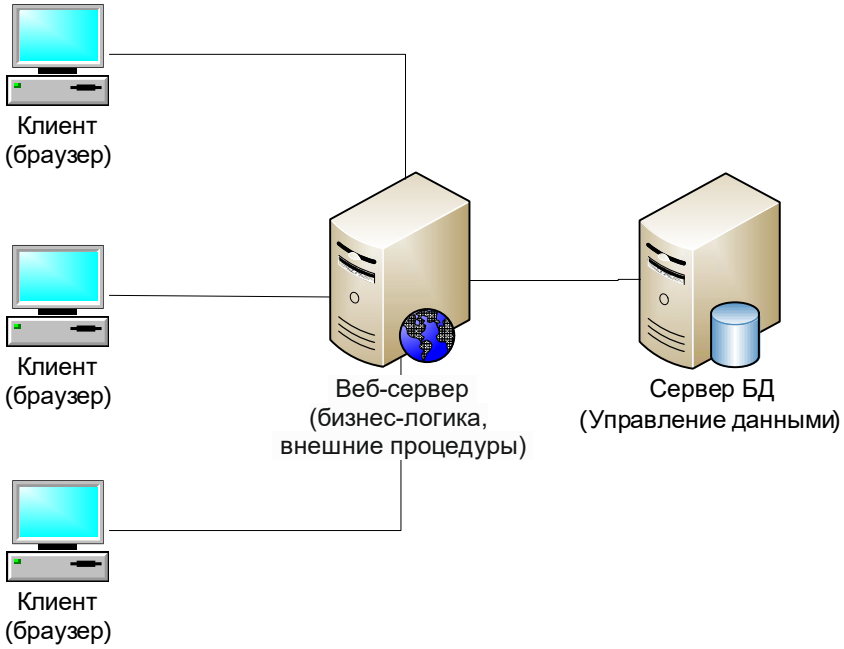


Рис. 4.6 – Архитектура веб-приложений

Особенности данного типа архитектуры: отсутствие необходимости использовать дополнительное ПО на стороне клиента (это позволяет реализовать клиентскую часть на всех платформах); возможность подключения практически

неограниченного количества клиентов; благодаря единственному месту хранения данных и наличия системы управления базами данных обеспечиваются минимальные требования для поддержания целостности данных; архитектура вебсистем не имеет существенных ограничений по объему базы данных.

Архитектура *распределенных систем* основана на предположении, что основной объем данных, необходимых пользователю, размещен на его персональном компьютере, обеспечив возможность его независимой работы (рис. 4.7). Поток исправлений и дополнений, создаваемый на этом компьютере, ничтожен по сравнению с объемом данных, используемых при этом. Поэтому если хранить непрерывно используемые данные на самих компьютерах и организовать обмен между ними исправлениями и дополнениями к хранящимся данным, то суммарный передаваемый трафик резко снизится. Это позволяет понизить требования к каналам связи между компьютерами и чаще использовать асинхронную связь, и благодаря этому создавать надежно функционирующие распределенные системы обработки данных.

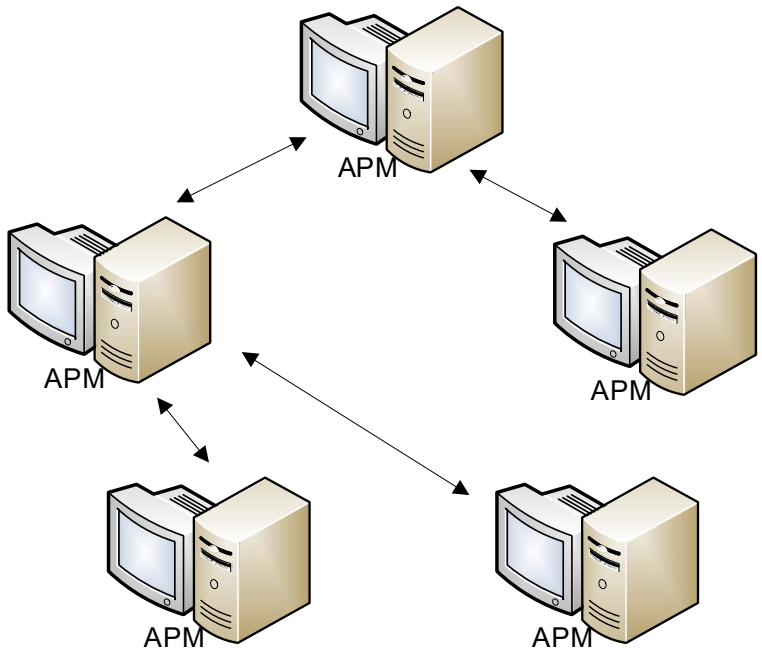


Рис. 4.7 – Архитектура распределенных систем

Так как данные, доступные на конкретном рабочем месте, находятся только на этом компьютере, при использовании средств шифрования и личных аппаратных ключей исключается доступ к данным посторонних, в том числе и ИТ-администраторов.

Реализация такой системы не элементарна и требует решения ряда проблем, одна из которых – своевременная синхронизация данных.

Сервис-ориентированная архитектура обеспечивает реализацию модульного подхода к разработке программного обеспечения, основанного на использовании сервисов со стандартизированными интерфейсами (рис. 4.8). Преимущество использования такого типа архитектуры заключается в том, что архитектура не привязана к какой-то определенной технологии используемой программной платформы и языков программирования; кроме того, чтоит отметить использование единообразных интерфейсов доступа к сервисам, организацию сервисов как слабосвязанных компонентов для построения систем.

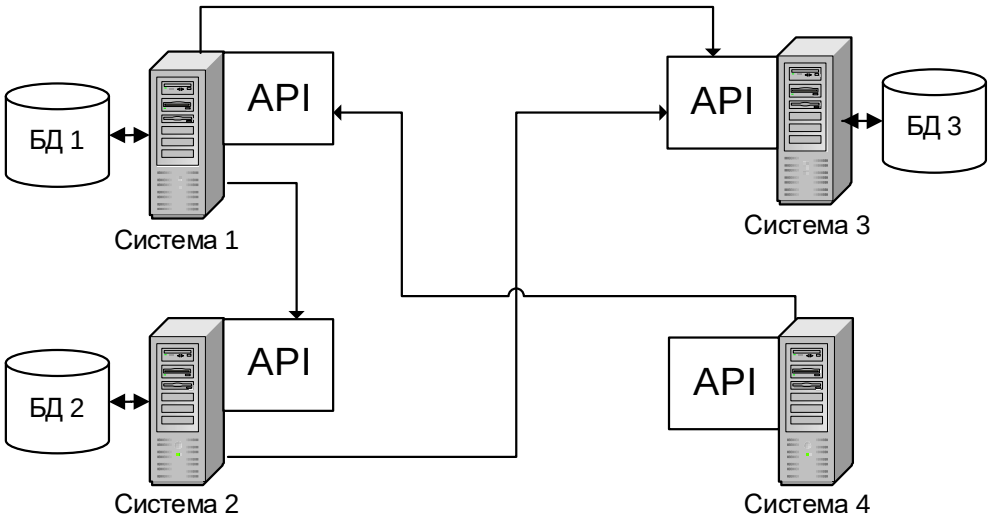
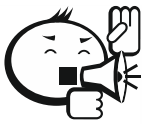


Рис. 4.8 – Сервис-ориентированная архитектура

4.3.3 Процессы проектирования программных продуктов



Процесс проектирования архитектурного или высокоуровневого дизайна включает следующие виды деятельности:

- разработку и документальное оформление архитектуры ПП, описывающей верхний уровень его структуры и идентифицирующей все программные компоненты последующих уровней;
- разработку и документальное оформление проекта верхнего уровня для внешних интерфейсов программного продукта и его интерфейсов с программными компонентами;
- разработку и документальное оформление проекта верхнего уровня для базы данных;

- разработку и документальное оформление предварительных версий пользовательской документации;
- определение и документирование требований к предварительному тестированию и графику работ по комплексированию программных компонентов и программного продукта в целом.

. . . . . .

Последним этапом проектирования архитектуры является оценивание детального проекта и требований к его тестированию по следующим критериям: взаимосвязи с требованиями к ПП; внешней согласованности с архитектурным проектом; внутренней согласованности между программными компонентами и программными модулями; соответствию выбранной модели жизненного цикла разработки и используемых стандартов методам проектирования; возможности тестирования программного проекта; влиянию на процессы функционирования и сопровождения ПП.

К ключевым вопросам детализированного проектирования относятся декомпозиция архитектуры на функциональные компоненты для независимого и параллельного их выполнения, принципы и механизмы их взаимодействия между собой, обеспечения качества и живучести программного обеспечения в целом. Один из вариантов декомпозиции описан в ГОСТ 34.003–90 «Информационная технология. Автоматизированные системы. Термины и определения», где предлагается выделять и описывать следующие элементы программного продукта:

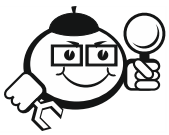
1) *интегрированный программный продукт* – совокупность двух и более программных продуктов, в которых функционирование одного из них зависит от результатов функционирования другого;
2) *программный продукт* – совокупность программных компонент, реализующих конкретный бизнес-процесс;
3) *сложный программный компонент* – совокупность программных кодов, реализующих сложную функцию бизнес-процесса;
4) *программный компонент* – совокупность программных кодов, реализующих элементарную функцию бизнес-процесса.

Такая декомпозиция на модули сложного программного обеспечения производится с целью получения более мелких и относительно независимых программных компонентов, каждый из которых несет различную функциональность. Содержание самих же процессов проектирования каждого элемента программного продукта аналогично процессам проектирования архитектурного дизайна.

4.3.4 Моделирование процессов разработки архитектуры программной системы управления и контроля работы скорой помощи

Для иллюстрации процесса проектирования программного продукта рассмотрим пример описания архитектуры программной системы управления и контроля работы скорой помощи с использованием диаграмм потоков данных [8].

На первом шаге выделим главные внешние сущности для системы: *звонящие* – люди, которые звонят, чтобы сообщить об экстренных ситуациях, и *машины скорой помощи*, работой которых управляет система.



Пример

Внешние сущности записи являются выходом системы, отражают требование законодательства и являются средством для измерения нефункционального требования «производительность системы» (рис. 4.9).

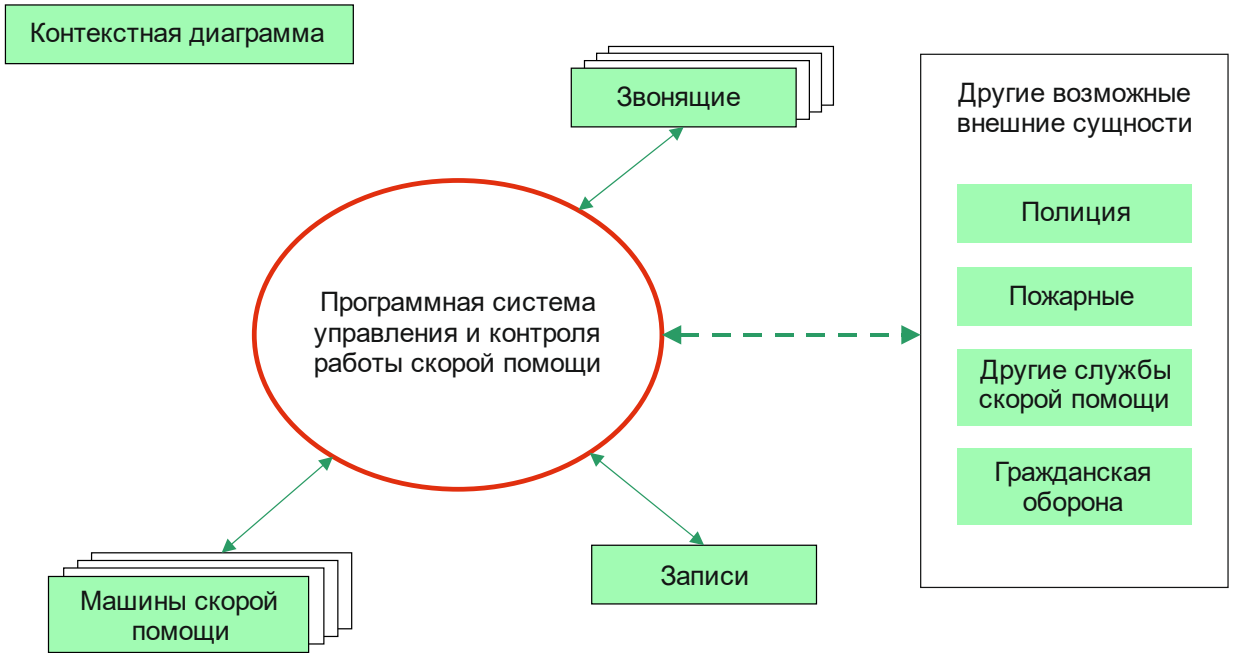


Рис. 4.9 – Контекстная диаграмма для системы управления и контроля скорой помощи

Другие возможные внешние сущности системы, представленные на диаграмме, тоже обязательны, но для простоты в данном примере не будут далее рассматриваться.

Следующим шагом является определение процессов – внутренних функций системы. Вначале для работы с каждой из внешних сущностей выделяют

функции верхнего уровня: обработка звонков, управление машинами скорой помощи, хранение записей, получая, таким образом, минимальную функциональную декомпозицию системы. После этого отображают основные данные, которые должны транслироваться между функциями верхнего уровня: текущие происшествия, состояния машин скорой помощи.

Пример

Далее функции верхнего уровня разбиваются на более мелкие составляющие (рис. 4.10). Полученная детализированная модель системы управления и контроля работы скорой помощи позволяет определить архитектуру программного продукта и на ее основе разработать перечень функциональных и нефункциональных требований к каждому из компонентов ПП (рис. 4.11).

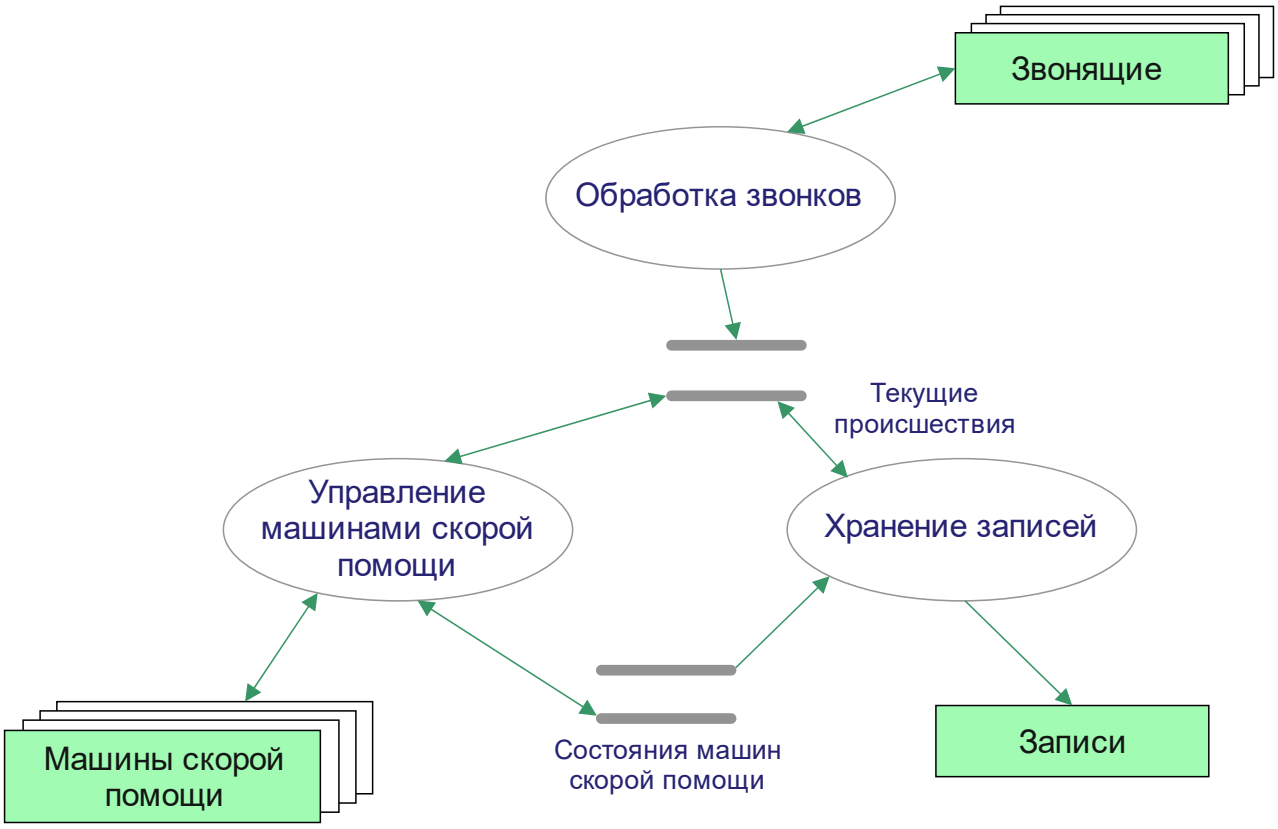


Рис. 4.10 – Модель системы управления и контроля скорой помощи

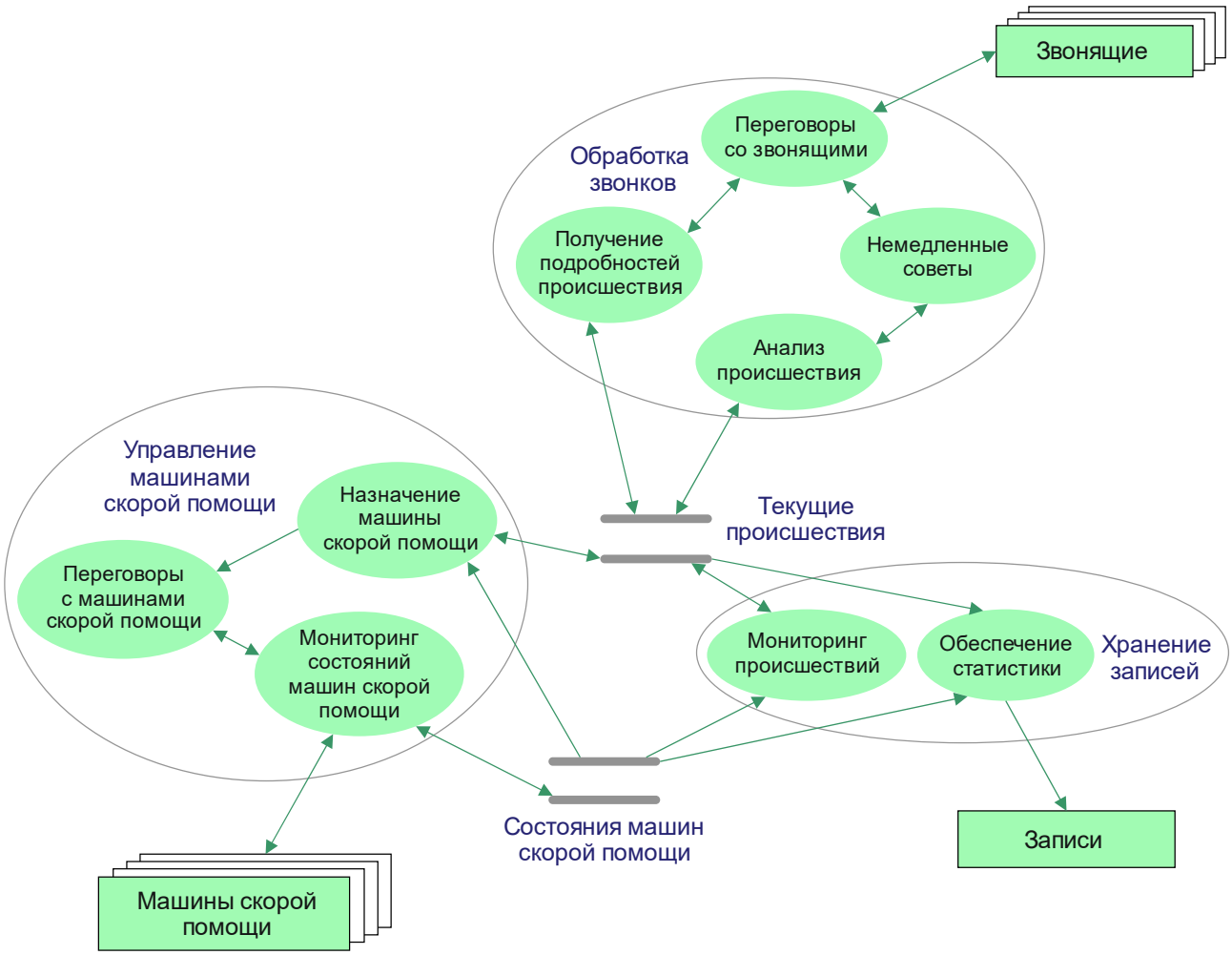
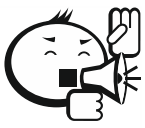
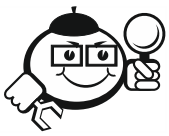


Рис. 4.11 – Детализированная модель системы управления и контроля работы скорой помощи



С учетом описанных выше фрагментов языка шаблонов пользовательское требование к реализации программного модуля (компонента) получения подробностей происшествия можно сформулировать в следующем виде: «*Диспетчер скорой помощи должен иметь возможность получать и отрабатывать информацию от звонящего о происшествии*». В свою очередь требование к программному модулю назначения машины скорой помощи можно записать как «*Диспетчер скорой помощи должен иметь возможность назначить машину для выезда на происшествие*».



Пример

Пример системного требования ко времени реакции на звонок может быть представлено в следующем виде: «*Система должна обеспечить диспетчеру* получение информации от звонящего до назначения машины скорой помощи в течение T минут» (рис. 4.12).

Рис. 4.12 – Архитектура ПП «Управление и контроль работы скорой помощи»

Таким образом, использование при моделировании диаграммы потоков данных позволяет разработчикам:

- отобразить потоки данных и функциональность системы;
- определить функции, потоки и хранилища данных;
- определить интерфейсы между функциями;
- реализовать процедуры получения системных требований;
- приступить непосредственно к разработке программного обеспечения.

4.4 Конструирование программного продукта

4.4.1 Процессы и инструментальные средства конструирования

Процесс конструирования программного обеспечения заключается в разработке исполняемых программных модулей посредством комбинации кодирования, верификации (*проверки*)*, модульного тестирования и комплексирования. Основные требования к процессам конструирования: минимизация сложности при создании программного кода, готовность* (*ожидание*) *периодических изменений требований, возможность проверки правильности программного кода, обязательное использование стандартов.*

Уменьшение сложности в конструировании программного обеспечения достигается при написании простого и легко читаемого кода, пусть и в ущерб стремлению сделать его идеальным (например, с точки зрения гибкости, размерности и т. д.). Большая часть программного обеспечения изменяется с течением времени, так как ПО не является изолированным от внешнего окружения. Более того, программное обеспечение является частью изменяющейся среды и должны меняться вместе с ней, а иногда и быть источником изменений самой среды.

«Конструирование для проверки» предполагает, что разработка программного обеспечения должна вестись таким образом, чтобы само программное обеспечение предоставляло возможность вести поиск причин сбоев, будучи прозрачным для применения различных методов проверки как на стадии независимого тестирования (например, инженерами-тестировщиками), так и в процессе эксплуатации, когда особенно важна возможность быстрого обнаружения и исправления возникающих ошибок, изменений в текущую версию ПО. Внесение изменений проводится с целью сохранения функциональной целостности системы на основе проведенного метрического анализа необходимости изменений программного кода.

Процессы конструирования должны проводиться с учетом требований внешних и внутренних стандартов. Внешние стандарты создаются разными российскими и международными организациями по стандартизации процессов жизненного цикла создания ПП, программных платформ, языков программирования, операционных сред,

систем управления базами данных, программно-технических интерфейсов и т. д. Внутренние стандарты поддерживают координацию между определенными видами деятельности, проектными группами и т. д.

. . . . . .

. . . . . .

При реализации процессов конструирования программных средств необходимо выполнять следующие виды деятельности:

1) *разработка и документальное оформление каждого программного модуля и базы данных, включая процедуры формирования исходных данных для тестирования каждого программного модуля и базы данных;*
2) *тестирование с использованием контрольных наборов тестов, для которых известен результат, и документальное оформление каждого программного модуля и базы данных;*
3) *внесение при необходимости изменений в документацию пользователя;*
4) *корректировка при необходимости требований к процедуре тестирования и определение графиков работ по комплексированию программных средств;*
5) *оценивание качества программного кода и документальное оформление результатов тестирования;*
6) *разработка и документальное оформление плана комплексирования по объединению программных компонентов и программных модулей в программный продукт;*
7) *проведение в соответствии с планом комплексирования программных компонентов и программных модулей в программный продукт, тестирование и документальное оформление результатов комплексирования и тестирования;*
8) *внесение изменений результатов комплексирования и тестирования в пользовательскую документацию* (*по мере необходимости*).

. . . . . .

Измерение эффективности и качества процессов конструирования ориентировано на количественную оценку объема кода, степени повторного использования кода (ПИК), вероятности появления дефектов и количественных показателей качества ПП.

К инструментальным средствам конструирования ПП относятся промышленные технологии проектирования и конструирования, в том числе интегрированные среды разработки программ (IDE), методы и технологии программирования.

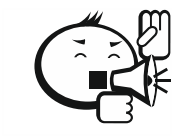
Промышленные технологии создания программного продукта (Microsoft Solution Framework (MSF), Rational Unified Process (RUP), DATARUN) основаны на использовании конкретных моделей разработки, содержат описания принципов, методов, применяемых процессов и операций, поддерживаются набором CASE-средств, охватывающих все этапы жизненного цикла продукта.

Интегрированные среды разработки программ (IDE) являются объединением программных средств, которые предназначены для написания (создания) программных продуктов. Как правило, интегрированные среды разработки поддерживает несколько языков программирования и включают: компилятор, интерпретатор, отладчик, средства автоматизации сборки, а также редактор текста.

Пример

1. VisualStudio разработан на C++ и C#, поддерживается Windows OS, включает различного рода дополнения, такие как:
   а) Re Sharper – поиск ошибок в коде во время написания кода программы, до компиляции;
   б) Subversion – систему контроля версий.
2. Komodo написана на JavaScript, XUL, Python, поддерживает десять языков программирования, среди которых присутствуют: PHP, Ruby, HTML5; работает на операционных системах: OS Linux, Windows и Mac OS.

Интегрированные среды разработки удобны для командной работы над проектами, постольку обеспечивают поддержку всех этапов ЖЦ создания программного обеспечения, позволяют сократить время на написание приложений, снизить затраты, повысить удобность разработки – что и является одной из основных целей программной инженерии.



К основным методам и технологиям разработки (написания кода) программного обеспечения относятся:

- процедурное программирование (язык С);
- системное программирование (Ассемблер);
- структурное программирование (С);
- логическое программирование (PROLOG);
- функциональное программирование (LISP, Haskell, Scala, Elm);
- визуальное программирование (Delphi);
- объектно-ориентированное программирование (Java, C#, C++, JavaScript, ActionScript, *Python, Ruby, PHP, Objective-C* и др.);
- экстремальное программирование как технология создания программного продукта в условиях ограниченного времени, неясных или быстро меняющихся требований;
- сервис-ориентированный подход к разработке ПП, основанный на использовании распределенных, слабо связанных компонентов, взаимодействующих по стандартизированным интерфейсам и протоколам;
- интеллектуальные многоагентные (мультиагентные) системы управления и поддержки групповой работы.

В процессе конструирования также должны использоваться внешние стандарты языков описания данных (XML, SQL и др.), средств коммуникации (COM, CORBA и др.), интерфейсов компонентов (POSIX, IDL, APL), описания сценариев UML и др.

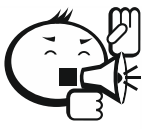
Командная работа над проектом объективно требует использования специального программного обеспечения, информационной поддержки процессов контроля и управления изменениями (управление версиями) исходных кодов и других артефактов разработки, используя общее хранилище. В зависимости от предназначения и инструментария системы контроля версий можно разделить на локальные, централизованные и распределенные.

Локальные системы контроля версий представлены простой базой данных, в которой хранятся все изменения нужных файлов. В *централизованных системах* контроля версий информация хранится на центральном сервере, где собраны все файлы текущей версии проекта: разработчики имеют возможность

скачивать копии файлов, а руководитель проекта организовать контроль текущего состояния работ. Однако эффективность использования централизованных систем контроля версий во многом зависит от надежности централизованного сервера. При использовании *распределенной системы* контроля версий разработчики имеют возможность не только выгружать последние версии файлов, но и полностью копировать весь репозиторий. В случае неисправности в работе сервера любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда разработчик скачивает последнюю версию файлов, он создает себе полную копию всех данных. Кроме того, в большей части этих систем можно работать с несколькими удаленными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах. На данный момент наиболее популярные системы контроля версий: RCS, CVS, Subversion, Aegis, Monoton, Git, Bazaar, Arch, Perforce, Mercurial, TFS.

4.4.2 Практические рекомендации по организации процессов конструирования при использовании промышленной технологии Microsoft Solution Framework

Процессы конструирования промышленной технологии Microsoft Solution Framework (MSF) ориентированы не просто на создание программного продукта, удовлетворяющего перечисленным требованиям, а на поиск решения проблем, стоящих перед заказчиком. Различие состоит в том, что перечисляемые заказчиком требования являются проявлениями некоторых более глубоких проблем и неточность, неполнота, изменение требований в процессе разработки – следствие недопонимания проблем. Поэтому в технологии MSF большое внимание уделяется анализу проблем заказчика и разработке вариантов системы для поиска решения этих проблем.



Рекомендации объединены в три раздела правил: «Выпустить», «Лучший проект», «Выпустить точно в срок» [6].

Раздел 1. «Выпустить»

1. *Вы не знаете того, чего вы не знаете*. В любом проекте всегда существуют неопределенности, но человеку свойственно отвергать свое незнание, подменяя истинное знание предположениями. Неизвестно, что является ошибкой, поэтому старайтесь быть критичными к себе. Найдите для своей команды мотивацию к тому, чтобы она постаралась обнаружить и понять максимум «белых пятен». Не пытайтесь обмануть себя предположениями.

2. *Старайтесь иметь четкое представление о состоянии проекта.* В проекте должны участвовать люди, способные объективно оценить готовность системы. Разработчики всегда более оптимистичны, чем тестировщики. Поощряйте плохие новости, и тогда снизится риск неожиданного провала в самом конце проекта. Самый лучший способ – полагаться на объективные данные, метрики (степень покрытия функциональности тестами, число активных дефектов, динамика их исправления и т. д.). Состояние осведомленности обязательно должно включать знание обо всех составляющих проекта и содержать точную информацию о статусе проекта в определенный период времени.

3. *Помните о треугольнике приоритетов.* Существуют три взаимосвязанных компонента любого проекта: ресурсы (люди и деньги), функциональность и сроки. Изменение одного из них всегда влияет на остальные. Можно зафиксировать значения только двух из этих показателей: например, зафиксировать характеристики времени и команды, чтобы получить требуемую функциональность; зафиксировать характеристики времени и команды, чтобы получить нужное время; зафиксировать характеристики времени и команды, чтобы получить требуемую численность команды разработки. Не забывайте, что не все задачи можно выполнять параллельно.

4. *Старайтесь быть на виду.* При планировании надо разбивать задачи на более мелкие, на выполнение которых требуется полдня или день. Тогда о том, что вы начинаете отставать, вы узнаете достаточно рано и сможете предпринять корректирующие шаги. Неделя для проекта разработки – это вечность. Каждый проект, который отстает на месяц, вначале отставал на один день. Узнайте об отставании прежде, чем наступит момент, когда исправить что-то будет уже невозможно. Несомненно, подобный стиль управления весьма опасен и периодически вас будут в нем обвинять. Но если вам удается довести до участников проекта понимание того, что его цель – предоставить готовый продукт в определенное время, ваши коллеги примут этот стиль.

5. *Используйте контрольные точки с отсутствием дефектов.* В каждом приложении можно выделить 20% функциональности, которая будет использоваться 80% времени; в свою очередь, в оставшихся 80% функциональности можно опять выделить 20% и т. д. Реализуйте сначала первые 20% функций, но полностью, а также протестируйте и соберите программу установки с прототипом документации. И только когда все это будет готово, двигайтесь дальше. Этот метод основан на идее выделения контрольных точек, в которых продукт должен содержать некое небольшое, но четко определенное количество дефектов. Их промежуточный контроль позволяет быстро понять, какие части проекта могут стать причиной проблем, и получить полную картину о проекте, а также дают возможность сфокусироваться на достижении целей каждой контрольной точки.

6. *Бойтесь разработчиков, сидящих в башнях из слоновой кости.* Разработчики должны уметь работать в команде, демонстрировать свой прогресс, делиться опытом и проверять то, что уже сделано. Избегайте «примадонн», которые считают себя умнее всех. Несомненно, проект лишь выиграет, если в команде появится пара гениев или просто отличных специалистов, но еще лучше будет, если их талант признает и команда, и все лица, заинтересованные в результатах проекта. Разработка программ осуществляется силами команды, для определения состава которой может пригодиться правило, принятое в Microsoft: шесть разработчиков, три инженера по качеству, один менеджер, два технических писателя. Это правило – результат многочисленных проб и ошибок, через которые прошла корпорация при разработке и масштабных платформ, и совсем небольших утилит.

7. *Плохая дата – не просто плохая дата*. Обычно вы заранее знаете, что опоздаете, знают это и все вокруг. Вы настаиваете на смене даты, но с каждой отсрочкой теряете доверие клиента. Вот хорошее правило: перенос даты должен происходить только в тех случаях, когда точно известны все составляющие и причины задержки. Изменение сроков требует ресурсов, поэтому следующая контрольная точка должна быть реалистичной. В противном случае подобный «проект» никогда не закончится.

8. *Сдвинув сроки, не проваливайте их*. Перенос срока – это симптом, который свидетельствует о выявлении «белых пятен». Такое случается весьма часто, поскольку разработка IT-решений является экспериментальным видом деятельности, в котором используются новые технологии. Все это ведет к неопределенности по поводу сроков проекта. Если сдвиг сроков выполнения проекта стал неожиданностью, значит, используемые методы общения с сотрудниками и

управления командой проекта надо менять. В то же время задержка дает возможность переоценить ресурсы и возможности продукта; в итоге это приносит положительный эффект. Сдвигая сроки, внимательно анализируйте ошибки и старайтесь их больше не допускать.

9. *Чем проще – тем лучше.* Лучше обещать меньше, но сделать больше, чем пообещать больше и не выполнить. Для заказчика важнее результат, а не обещания. А для успеха проекта стоит выбирать максимально простые, но надежные решения.

10. *Время для проектирования – это время для проектирования.* Оценивая способы проектирования, всегда учитывайте временной фактор и выбирайте из альтернативных решений наименее рискованное и оптимальное по времени реализации. Часто в погоне за красотой реализации проектировщик склонен выбрать вариант, совершенно не укладывающийся в сроки проекта. Помните, что время – весьма ограниченный ресурс, который дан для того, чтобы достичь успеха, а не удивить.

11. *Если вы не можете что-то собрать, значит, вы не сможете это выпустить.* Продукт должен постоянно собираться (компилироваться вместе со всеми компонентами системы, документацией и программой установки). Это необходимо потому, что с самого начала нужно определить все составляющие будущего продукта и адекватно оценить степень их готовности. В противном случае вы обязательно что-нибудь забудете, и это станет весьма неприятной неожиданностью, особенно перед окончанием работ. Пусть на начальном этапе это будут скромные зачатки будущей системы, придет время, и из них вырастет отличный продукт. С другой стороны, промежуточная версия продукта – хороший способ продемонстрировать заказчику факт готовности того или иного фрагмента работы.

12. *Мысли о многоплатформенности.* Старайтесь «не перебарщивать» с числом платформ, на которых будет работать система. Держите в голове тот факт, что разработка для каждой из платформ зачастую представляет собой переработку большей части функциональности, а также ее тщательное тестирование и последующее исправление ошибок. Соответственно, это увеличивает как стоимость системы, так и затраты на будущую поддержку. Помните, что пользователям чаще всего нужен продукт, а не его удивительная гибкость.

Раздел 2. «Лучший продукт»

1. *Заказчик – это ваше все*. Старайтесь в следующих версиях учитывать даже весьма странные, нечетко выраженные пожелания клиентов. Часто в этих

требованиях скрывается то, что делает продукт уникальным и отличимым от других, добавляет ему маркетинговой привлекательности и заставляет пользователей использовать его долгие годы. Помните, что заказчик лучше вас знает, что ему нужно.

2. *Самое главное – единство и интеграция*. Единство причины и единство исполнения должны стать девизами команды разработчиков.

3. *Двигайтесь правильным курсом*. Цель – основная идея вашей разработки. Все оценки продукта основываются именно на ней, поэтому она должна быть очень четкой. Старайтесь как можно раньше наметить цель и сохранить веру в нее вплоть до конца проекта.

4. *Будьте гибким*. Часто по ходу проекта требования к системе могут изменяться – будьте готовы к этому. Старайтесь постоянно проверять, насколько мнение пользователя соответствует поставленной цели. Используйте для этого промежуточные версии продукта, вовлекая заказчика в процесс работы с системой как можно раньше. Однако, собираясь менять курс, помните: цель должна остаться прежней.

5. *Соблюдайте баланс*. Правильно расставьте акценты на разных составляющих проекта. Ни в коем случае не увлекайтесь наращиванием свойств какой-либо из возможностей продукта – это станет причиной дискомфорта пользователей и может привести к серьезным проблемам со сроками реализации.

6. *Развивайте продукт постепенно*. Правильное развитие выглядит так: ранние стадии разработки определяют более поздние, ошибки не повторяются, а результат отвечает потребностям конечного пользователя. Плавное развитие вселяет в вас ощущение предсказуемости и стабильности процесса разработки.

7. *Продукт – это иерархия компонентов*. Следуя этому принципу, элементам проекта уделяют внимание пропорционально их важности, что обеспечивает стабильность и сбалансированное развитие. Иерархию очень удобно использовать как скелет для постепенного расширения системы, сначала вы реализуете основу, а затем наполняете ее будущим содержимым; новые компоненты опираются на уже разработанные.

8. *Все должны разделять общее видение продукта*. Все члены команды должны знать, какие цели должны быть достигнуты, как продукт должен выглядеть, какова стратегия его разработки. Если в команде появляются противники текущей цели, постарайтесь дать им слово и аргументировать свое видение, быть может, оно лишь улучшит будущую систему. Все противоречия должны быть разрешены, а видение предмета приведено к единству.

Раздел 3. «Выпустить точно в срок»

Ваша главная задача – выпустить продукт. Помните:

- команда обязана поставить продукт в срок, а все члены команды должны верить в то, что это возможно;
- каждый должен понимать, что от него для этого требуется, а менеджер должен сделать все от него зависящее, чтобы иметь все шансы сделать то, что требуется;
- любой должен не просто хотеть, а гореть желанием достичь цели.

Выпуск продукта должен стать целью каждого члена команды, самым ожидаемым событием для всех!

4.5 Тестирование программного продукта

В соответствии с принятой терминологией базовым понятием тестирования является «тест», который выполняется в заданных условиях и на проверочных наборах данных. В процессе тестирования выявляются следующие недостатки: отказы и дефекты, сбои, ошибки. Важно четко разделять *причину* нарушения работы прикладных программ, обычно описываемую терминами *недостаток* или *дефект*, и наблюдаемый нежелательный эффект, вызываемый этими причинами, – *сбой*. *Тестирование, ориентированное на дефекты*, направлено на обнаружение наиболее вероятных ошибок, предсказываемых заранее, например, в результате анализа возможных рисков программного проекта. Термин *ошибка*, в зависимости от контекста, может описывать и как причину сбоя, и как сам сбой. Тестирование позволяет обнаружить дефекты, приводящие к сбоям.

В соответствии с принятой архитектурой ПП выделяют следующие уровни тестирования [11]:

- *модульное тестирование*, предполагающее проверку отдельных, изолированных и независимых элементов (модулей, компонентов);
- *интеграционное тестирование*, которое ориентировано на проверку связей и способов взаимодействия (интерфейсов) компонентов друг с другом;

- *тестирование программного обеспечения*, предназначенное для проверки правильности функционирования в целом, с обнаружением отказов и дефектов и их устранением.

. . . . . .

При этом контролируется и выполнение нефункциональных требований (безопасность, надежность и т. д.), а также правильность задания и выполнения внешних интерфейсов со средой окружения.

. . . . . .

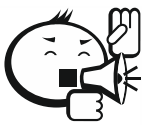
Для проверки качества и работоспособности ПП выделяют следующие виды тестирования [11]:

1) *функциональное тестирование*, которое заключается в проверке соответствия выполнения функциональных возможностей ПП;
2) *регрессионное тестирование* – тестирование программного обеспечения или его компонентов после внесения в них изменений;
3) *тестирование эффективности функционирования* – проверка в соответствии со спецификациями требований производительности, пропускной способности, максимального объема данных, системных ограничений и т. д.;
4) *нагрузочное* (*стресс*) *тестирование* – проверка поведения ПП при максимально допустимой нагрузке;
5) *внутреннее* (*альфа*) *и внешнее* (*бета*) *тестирование* – без плана тестирования и с планом тестирования соответственно;
6) *тестирование конфигурации* – проверка структуры и идентификации системы на различных наборах данных, а также проверка работы системы в различных конфигурациях;
7) *приемочное тестирование* – проверка в соответствии с заранее подготовленной программой и методикой испытаний поведения системы на этапе приемки-сдачи ее заказчику.

. . . . . .

В зависимости от этапа жизненного цикла выделяют альфа-тестирование и бета-тестирование.

. . . . . .



Альфа-тестирование – этап начала тестирования ПП специалистами-тестерами. Цель данного этапа – проверка достижения необходимого качества функционирования ПП. Чаще всего альфа-тестирование проводится на ранних стадиях процесса разработки ПП,

но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приемочного тестирования.

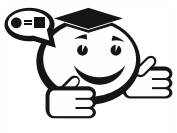
Бета-тестирование проводится с привлечением потенциальных пользователей продукта. Целью данного этапа является оценка качества ПП и получение информации о продукте и соответствующей ему документации от его будущих пользователей.

. . . . . .

Для измерения процессов и результатов тестирования используются соответствующие метрики тестирования. Измерение результатов тестирования касается оценки качества получаемого продукта. *Оценка программ в процессе тестирования* базируется на размере программ (например, в терминах количества строк кода или функциональных точек) или их структуре (например, с точки зрения оценки ее сложности в тех или иных архитектурных терминах). Структурные измерения могут также включать частоту обращений одних модулей программы к другим. *Эффективность тестирования* может быть измерена путем определения типов дефектов, которые могут быть найдены в процессе тестирования, и изменения их частоты во времени. Эта информация позволяет прогнозировать качество ПП и совершенствовать в дальнейшем процесс разработки в целом. Тестируемая программа может оцениваться также *на основе подсчета и классификации найденных дефектов*. Для каждого класса дефектов можно определить отношение между количеством соответствующих дефектов и размером программы.

В ряде случаев процесс тестирования требует систематического выполнения тестов для определенных набора элементов программы, задаваемых ее архитектурой или спецификацией. Соответствующие метрики позволяют оценить степень охвата характеристик системы и глубину их детализации. Такие метрики помогают прогнозировать вероятностное достижение заданных параметров качества системы.

Документация на тестирование включает описание тестовых документов, их связи между собой и с процессом тестирования. Без документации на процессы тестирования невозможно провести сертификацию и оценку зрелости программного продукта. После завершения тестирования рассматриваются вопросы стоимости и рисков, связанных с появлением сбоев и недостаточно надежной работой системы. Стоимость тестирования является одним из ограничений, на основе которого принимается решение о прекращении или продолжении тестирования.



Процессы тестирования ПП заключаются в проверке работоспособности и сравнении полученных результатов с целевыми показателями качества, заложенными в техническом задании.

При этом сами процессы тестирования должны быть организованы с учетом следующих положений:

1) все работы и результаты процесса тестирования должны обязательно фиксироваться;
2) форма журналирования работ и их результатов должна быть такой, чтобы соответствующее содержание было понятно, однозначно интерпретируемо и повторяемо другими лицами;
3) тестирование должно проводиться в соответствии с заданными и документированными процедурами;
4) тестирование должно производиться над однозначно идентифицируемой версией и конфигурацией ПП.

Положительные результаты тестирования являются подтверждением того, что скомплексированный программный продукт полностью удовлетворяет установленным требованиям.

При реализации процесса необходимо выполнять следующие виды деятельности:

1) планирование работ по тестированию (составление планов, тестов, наборов данных) и измерению показателей качества ПП;
2) генерация необходимых тестовых сценариев, соответствующих среде выполнения ПП;
3) квалификационное тестирование на соответствие тестам и тестовым сценариям;
4) сбор данных об отказах, ошибках и других непредвиденных ситуациях при выполнении программного продукта;
5) подготовка отчетов и оценка результатов тестирования и пользовательской документации по следующим критериям: тестовому покрытию требований к ПП; соответствию ожидаемым результатам; реализуемости этапа комплексирования ПП и дальнейшего тестирования; влиянию на процессы функционирования и сопровождения программного продукта;

6) внесение изменений по мере необходимости в пользовательскую документацию;
7) подготовка ПП для комплексирования с программно-аппаратной средой эксплуатации, системного тестирования, инсталляции и приемки-сдачи.

Эффективность и качество результатов тестирования зависят от используемых инструментальных средств автоматизации процессов тестирования ПП. На рынке существует множество как коммерческих инструментальных средств, так и инструментальных средств с открытым исходным кодом: Selenium, Katalon Studio, Unified Functional Testing, TestComplete, Robot Framework т. д.

В заключение следует отметить, что достаточно сложно определить границы между проектированием, конструированием и тестированием, так как все они связаны в единый комплекс процессов жизненного цикла и в зависимости от выбранной модели разработки цикла и применяемых методов (методологий) такое разделение может выглядеть по-разному.

Контрольные вопросы по главе 4

1. Поясните содержание стандарта «Единая система программной документации» (ЕСПД): ГОСТ 19.102–77 ЕСПД «Стадии разработки».
2. Поясните содержание стандарта IEEE-1074–1997 «Процессы и действия жизненного цикла программного обеспечения».
3. Раскройте содержание бизнес-требований, пользовательских требований, функциональных и нефункциональных требований по классификации К. Вигерса.
4. Перечислите и прокомментируйте виды деятельности по работе с требованиями.
5. Раскройте содержание этапа проектирования архитектуры программного обеспечения.
6. Перечислите и раскройте содержание типовых архитектур программных систем.
7. Раскройте содержание этапа конструирования программного продукта.
8. Раскройте содержание этапа тестирования программного продукта.

5 Жизненный цикл вывода на рынок программного продукта

5.1 Ввод в эксплуатацию и сопровождение программного продукта

После устранения выявленных в процессе тестирования недочетов осуществляется релиз – выпуск окончательной версии ПП, готового для ввода в эксплуатацию. Ввод в эксплуатацию начинается с процессов инсталляции программного продукта и организации приемки-сдачи.

Процесс инсталляции заключается в установке программного продукта, удовлетворяющего заданным требованиям, в программно-аппаратную среду применения.

Если инсталлируемый ПП заменяет существующую программную систему, то в этот период времени необходимо предусмотреть поддержку необходимых параллельно выполняемых процессов. По мере готовности ПП устанавливается в среде применения для проведения приемо-сдаточных испытаний.

Процесс приемки-сдачи системы заключается в подтверждении того, что инсталлированный программный продукт соответствует заданным в техническом задании требованиям.

В процессе реализации приемки-сдачи разработчик должен:

- установить программный продукт в программно-аппаратную среду применения;
- обеспечить работоспособность программного продукта при приемо-сдаточных испытаниях;
- провести обучение сотрудников заказчика в процессе проведения испытаний.

Результаты процесса приемки программных средств должны быть документированы. Проблемы, обнаруженные в течение приемки-сдачи, идентифицируются и передаются разработчику (поставщику) для доработки.

Ввод в эксплуатацию и сопровождение ПП начинается сразу после подписания положительного заключения о приемке-сдаче и документального подтверждения начала использования программного продукта. Стадия эксплуатации продолжается до полного изъятия продукта из эксплуатации, что связано со спецификой использования программного продукта.

Сопровождение ПП начинается одновременно с эксплуатацией ПП, составляет большую часть ЖЦ программного продукта (по некоторым оценкам составляет порядка 67% времени ЖЦ ПП) и действует в течение периода гарантии или технической поддержки ПП, осуществляемой на договорной основе.

Объективная потребность в сопровождении связана:

- с обнаружением при эксплуатации ПП скрытых дефектов и необходимостью устранения сбоев;
- улучшением характеристик качества программного продукта;
- реализацией новых функциональных возможностей;
- созданием интерфейсов взаимодействия с внешними системами;
- адаптацией для обеспечения возможности работы ПП на другой программно-аппаратной платформе;
- изменением бизнес-процессов в предметной области.

Согласно действующим стандартам процесс сопровождения ПП заключается в обеспечении эффективной по затратам поддержки функционирования и модификации программного продукта (*исправление обнаруженных ошибок, повышение производительности продукта и адаптация к изменившимся условиям работы или требованиям*)*, обучения и консультирования пользователей в режиме горячей линии.*

В зависимости от исходных условий состояния ПП в [12] выделяется четыре категории сопровождения.

1. Корректирующее сопровождение: «реактивная» модификация программного продукта, выполняемая уже после передачи в эксплуатацию для устранения сбоев.
2. Адаптирующее сопровождение: модификация программного продукта на этапе эксплуатации для обеспечения продолжения его

использования с заданной эффективностью (с точки зрения удовлетворения потребностей пользователей) в изменившемся или находящемся в процессе изменения бизнес-окружения, порождающее новые требования к ПП.

3. Совершенствующее сопровождение: модификация программного продукта на этапе эксплуатации для повышения характеристик качества (производительности, удобства сопровождения и т. д.).
4. Профилактическое сопровождение: модификация программного продукта на этапе эксплуатации для идентификации и предотвращения скрытых дефектов до того, когда они приведут к реальным сбоям.

. . . . . .

Для организации процесса сопровождения в компании-разработчике создаются специальные подразделения по оказанию технической поддержки конечным пользователям, обычно именуемые *службой поддержки пользователей*, основными функциями которых являются:

- получение и обработка запросов пользователей, выяснение первопричины, которая привела к проблеме;
- регистрация и мониторинг отчетов о проблемах и заявках от пользователей и обеспечение обратной связи с ними;
- определение типа запроса: неисправность в работе ПП, предложения по развитию и модификации;
- формирование ответов – рекомендаций пользователям (в случае типовой проблемы);
- передача запросов пользователей в отдел разработки для определения программных модулей и документации, нуждающихся в модификации (в нетипичных ситуациях);
- анализ отчетов о проблемах и/или заявках на модификацию программных средств с учетом масштабов модификации, требуемых финансовых средств, времени на модификацию, критичности модификации, воздействия ее результатов на эксплуатационные характеристики, безопасность или защищенность ПП.

5.2 Продвижение тиражного программного продукта на рынок

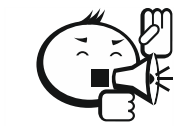
Коммерциализация тиражных программных продуктов предполагает проведения комплекса маркетинговых мероприятий (*маркетинговых коммуникаций*) *с целью доведения до целевой аудитории потребителей информации о существовании продукта* (*и самой компании*) *и конкурентных преимуществах использования продукта, что в конечном счете должно стимулировать потребителя к апробации продукта и принятию решения о его приобретении.*

Все эти вопросы рассматриваются при планировании и организации исполнения программы продвижения ПП на рынок [12].

Исходными данными для продвижения ПП на рынок является маркетинговая цель компании-разработчика – желаемый результат, который должен быть получен в результате реализации программы продвижения ПП в определенном интервале времени при ограничениях на имеющиеся ресурсы компании-разработчика. Маркетинговая цель может выражаться в денежных или натуральных показателях, непосредственно связанных с продажами ПП (достижение определенного объема продаж, увеличение доли рынка, привлечение новых клиентов, увеличение прибыли и др.).

Очевидно, что малая компания на стадии вывода ПП не сможет охватить весь рынок потенциальных потребителей в силу ограниченности собственных ресурсов, существующей дифференциации потребностей потребителей, их инфраструктуры и способов организации закупок. В этой связи на первом этапе необходимо осуществить сегментирование рынка, т. е. разделить всю совокупность потребителей на группы со схожими характеристиками и потребительскими предпочтениями.

Потребителями промышленного рынка ПП являются предприятия и организации, разделенные по географическому положению, отраслевой принадлежности, форме собственности и т. д.



С точки зрения потребительского поведения потенциальных пользователей промышленного рынка решение о приобретении ПП принимают следующие специалисты: непосредственные пользователи программного продукта; специалисты IT-служб, отвечающие за

установку, адаптацию и техническую поддержку программного продукта; первые руководители компании.

При разделении потребительского рынка потенциальных пользователей ПП предлагается использовать две переменные сегментирования: разделение потребителей *по полу и возрасту* и *по особенностям потребительского поведения*. Например, потенциальные потребители по половозрастным характеристикам: юноши 9–14 лет; мужчины 15–24 лет; девочки 9–14 лет; девушки 15–24 лет; женщины 25–40 лет; женщины старше 40. С точки зрения поведения потенциальных пользователей потребительского рынка их можно распределить по четырем типам: первопроходцы, прагматики, консерваторы, копуши.

. . . . . .

Для успешной сегментации рынка целесообразно применять пять принципов [4]:

- *различия между сегментами*, т. е. в результате проведения сегментации должны быть получены различающиеся группы потребителей;
- *сходство потребителей*, т. е. однородность потребителей в сегменте по совокупности показателей, значимых для определения покупательских предпочтений;
- *оптимальное количество потребителей*, иными словами, сегменты должны быть достаточно большими для обеспечения необходимого уровня продаж;
- *измеримость характеристик потребителей* или наличие качественных либо количественных параметров потребителей для формирования оптимальных стратегий позиционирования продукта;
- *достижимость потребителя* подразумевает наличие каналов коммуникации компании-разработчика с потребителями сегмента.

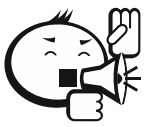
Далее следует выбрать один или несколько наиболее перспективных сегментов рынка (целевых сегментов), провести бизнес-анализ, основанный на интегральной оценке привлекательности сегмента (ожидаемых прибыли и затрат) в зависимости от комплектности поставки ПП. По результатам бизнес-анализа выбираются наиболее перспективные (целевые) сегменты, обслуживание которых соответствует маркетинговой цели компании и возможностям компании-разработчика.

Для каждого целевого сегмента важно осуществить выбор стратегии *позиционирования продукта*, простого и четкого утверждения, объясняющего, почему целевой аудитории следует приобретать и использовать именно этот ПП, и реализовать комплекс маркетинговых мероприятий, благодаря которым данный продукт по отношению к конкурирующим продуктам занимает собственное, отличное от других и выгодное для компании место. У потенциальных потребителей должно сложиться единое мнение о преимуществе продукта компании по сравнению с аналогичной продукцией конкурентов. В этой связи при разработке стратегии позиционирования прежде всего необходимо четко определить группу пользователей на целевом рынке, для которых предназначен предлагаемый продукт, и выделить их потребительские предпочтения и проблемы, которые потенциальный потребитель может решить при внедрении продукта.

Практическая реализация стратегии позиционирования предполагает разработку и реализацию комплекса мероприятий по коммуникационному взаимодействию с представителями целевой аудитории, включая:

- разработку структуры и содержания рекламного коммуникационного сообщения (КС);
- выбор канала коммуникаций с целевой аудиторией;
- выбор инструментов, мест и продолжительности размещения коммуникационных сообщений.

Содержание рекламного КС должно не только информировать целевую аудиторию о существовании ПП и компании-разработчика, но и вызывать интерес у определенных групп специалистов компании-потребителя, а также стимулировать у последних принятие решения об апробации и покупке ПП. Учитывая специфику ПП как нематериального (цифрового) продукта, в качестве канала распространении информации о программных продуктах целесообразно использовать Интернет с применением инструментария интернет-маркетинга и электронной коммерции. При этом могут использоваться различные инструменты интернет-маркетинга, в частности: медийная реклама, контекстная реклама, интернет-PR, участие в партнерских программах, продвижение в социальных медиа, поисковая оптимизация, адресная рассылка рекламных материалов. Выбор конкретных инструментов для продвижения ПП и мест размещения КС должен основываться на анализе определенных коммуникационных и стоимостных характеристик. В результате планирования необходимо получить план размещения КС, который четко показывает, где и как долго они будут размещаться, сколько это стоит и достижения каких результатов возможно ожидать.



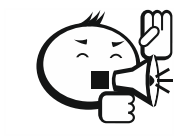
В результате реализации представленных стадий формируется программа продвижения ПП, которая содержит комплекс решений, включающий:

- формальное описание базового рынка ПП;
- перечень целевых сегментов и вариантов тиражирования ПП в них, включая ожидаемые объемы продаж и требуемых инвестиций для продвижения ПП;
- план размещения КС, содержащий прогнозные показатели достижения желаемых ответных реакций представителей целевой аудитории;
- набор КС, ориентированных на удовлетворение потребительских предпочтений различных групп пользователей.

В процессе реализации программы продвижения необходимо спроектировать процессы и настроить механизмы мониторинга результативности и оценки эффективности программы продвижения. В основу этих процессов должен быть положен анализ степени достижения плановых количественных коммуникационных и стоимостных показателей, конверсии достижения ожидаемой ответной реакции. Эти сведения позволят в случае необходимости в процессе реализации программы продвижения скорректировать план размещения КС, перераспределив бюджет между используемыми инструментами, рекламными площадками и местами размещения КС. Собранные по итогам реализации программы продвижения метрики позволят оценить как эффективность программы продвижения, так и достижение поставленной маркетинговой цели. Итоговый анализ эффективности реализации программы осуществляется с целью накопления эмпирических данных, служащих источником для прогнозных расчетов при последующем планировании вывода ПП на новые сегменты рынка или организации продвижения нового ПП.

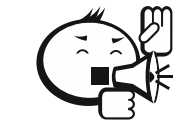
5.3 Жизненный цикл фазы вывода на рынок тиражного программного продукта

Жизненный цикл фазы вывода на рынок начинается с первой продажи ПП и с маркетинговой точки зрения состоит из четырех стадий [12].



На стадии *роста рынка* объем продаж начинает стремительно расти, новый продукт пользуется спросом со стороны пользователей. Первые пользователи продолжают эксплуатацию, новые покупатели начинают следовать их примеру, особенно если они слышат хорошие отзывы.

Процессы продвижения нацелены на информирование максимального количества представителей целевой аудитории, выход на новые сегменты рынка, информирование существующих пользователей об изменениях или доработках продукта. Поскольку расходы на продвижение соотносятся со все большим объемом продаж, издержек на производство ПП фактически нет, а сопровождение обычно оговаривается отдельными соглашениями с пользователями, следует ожидать роста прибыли. Вместе с тем в силу увеличения числа пользователей возможно значительное увеличение затрат на сопровождение продукта.



Стадия *зрелости рынка* обычно более длительна по сравнению с предыдущей стадией, характеризуется замедлением роста продаж ПП и включает три этапа:

- *растущая зрелость* – объем продаж медленно увеличивается, так как на рынке появляются пользователи, принявшие решение о покупке с некоторым опозданием, хотя в основном спрос обеспечивают постоянные пользователи;
- *стабильная зрелость* – объем продаж находится на постоянном уровне и обеспечивается главным образом продлением лицензий на использование ПП;
- *снижающаяся зрелость* – объем продаж начинает снижаться, поскольку некоторые пользователи начинают использовать новые ПП.

На данной фазе особое значение придается процессам мониторинга за состоянием рынка. Основным результатом этих процессов является принятие решений относительно дальнейшей маркетинговой стратегии, которая позволит максимально продлить фазу зрелости. Возможно использование трех стратегий: модификация рынка, модификация продукта и модификация комплекса маркетинговых мероприятий.

Модификация рынка заключается в разработке дополнительных мероприятий, направленных на увеличение числа пользователей ПП и вход на новые сегменты рынка.

Модификация продукта нацелена на улучшение как функциональных, так и нефункциональных характеристик ПП. Придавая продукту новые свойства, компания зарабатывает репутацию инноватора и закрепляет лояльность целевых сегментов, для которых эти новые свойства считаются важными. Основной недостаток стратегии улучшения свойств заключается в том, что новые свойства легко копируются конкурентами, и если компания не будет постоянно стремиться к лидерству, одноразовая модификация продукта вряд ли окупится в долгосрочной перспективе.

Модификация комплекса маркетинговых мероприятий заключается в изменении одного или нескольких элементов маркетинга: цены, распределения, рекламы, стимулирования сбыта, прямых продаж, сопровождения. На этом этапе стимулирование сбыта оказывает большее воздействие на потребителей, поскольку они утвердились в своих привычках и предпочтениях, а психологическое воздействие (реклама) не столь эффективно, как финансовое (стимулирование сбыта).

С точки зрения маркетинга стадия *упадка рынка* в жизненном цикле продукта характеризуется падением спроса на ПП и, как следствие, снижением объема продаж и прибыли. Упадок может быть обусловлен разными причинами: устареванием продукта в связи с развитием технологий, появлением более сильных конкурентов, изменением предпочтений потребителей.

Модернизация продукта, снижение цен, увеличение затрат на маркетинг могут только продлить эту стадию. На данной стадии компании необходимо принять решение о сроках прекращения тиражирования, длительности сопровождения продукта, возможности разработки новой версии ПП. Если у компании уже имеется новая версия ПП, самое время стимулировать имеющихся пользователей к ее использованию, однако эти мероприятия лучше планировать в отдельной программе продвижения новой версии ПП.

Главной особенностью процессов сопровождения на данной стадии является оказание услуг по поддержке работоспособности ПП (без проведения модификации и обновления ПП) в объеме, необходимом для его нормальной работы до утилизации и/или перехода на использование новой версии.

На стадии *вывода программного продукта из эксплуатации* прекращаются процессы сопровождения ПП, проводится изъятие из эксплуатации программного продукта и связанных с ним программных подсистем. Все связанные с выводимым из эксплуатации программным продуктом подсистемы, документы и данные должны быть помещены в архивы.

При планировании фазы вывода из эксплуатации необходимо определить сроки вывода продукта из эксплуатации, способы информирования пользователей, процедуру вывода продукта из эксплуатации, наличие других продуктов, которые можно предложить пользователю взамен выводимого из эксплуатации.

Контрольные вопросы по главе 5

1. Раскройте содержание этапа ввода в эксплуатацию и сопровождения программного продукта.
2. Перечислите и прокомментируйте переменные сегментирования промышленного и потребительского рынков на этапе продвижения программного продукта.
3. Прокомментируйте содержание этапов роста и зрелости рынка программных продуктов.
4. Прокомментируйте содержание этапов упадка рынка и вывода программных продуктов с рынка.

6 Управление программными проектами

6.1 Основные понятия и определения

Классическая теория управления проектами основана на двух базовых понятиях: проекта как совокупности работ по получению уникального результата, оформленных в виде плана действий, и продукта как конечного результата реализации проекта. Остановимся на этих понятиях более подробно.

В [7] *приводятся следующие определения проекта:*

1) *проект – временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов;*
2) *проект – комплекс взаимосвязанных действий, предпринимаемых с целью получения конкретных уникальных результатов при заданных ограничениях по времени, денежным средствам, ресурсам и качеству конечных результатов проекта;*
3) *проект – комплекс взаимосвязанных действий или задач, имеющих определенную цель, которая будет достигнута в рамках выполнения некоторых заданий, характеризующихся определенными датами начала и окончания, пределами финансирования и ресурсами.*

Анализ представленных определений позволяет выделить следующие *характеристики* проекта:

- направленность проекта на достижение конкретного конечного результата, определяемого в терминах требуемых ресурсов, качества и времени реализации;
- уникальность проекта как разового (неповторяющегося) мероприятия, требующего специфической организации управления;
- ограниченность проекта по времени и ресурсам (финансовым, трудовым, материальным) и как следствие – необходимость нахождения постоянного компромисса между объемом работ, ресурсами, временем, качеством и рисками и их перераспределения в ходе выполнения проекта;

- структурная сложность проекта как комплекса тесно взаимосвязанных мероприятий и его высокая неопределенность, обусловленная возможными изменениями условий реализации, потребности в тех или иных видах ресурсов.

Основываясь на вышеизложенном, ***программный проект*** *можно определить как* «*комплекс взаимосвязанных работ, выполняемых командой проекта с целью получения уникального программного продукта или услуги в течение заданного периода при установленном бюджете и потребляемых в ходе реализации проекта ресурсах в условиях повышенного риска, требующих специфического управления*».

В свою очередь конечным результатом реализации программного проекта является ***программный продукт****, который можно определить как* «*совокупность записанных на носителях данных программных компонентов, являющихся продуктом промышленного производства, предназначенных для поставки, передачи или продажи пользователю, снабженных технической документацией, рекламными материалами, инструкциями по обучению пользователей, гарантийными обязательствами по сопровождению и обслуживанию*».

*Управление программным проекто*м следует рассматривать как деятельность по планированию, организации, мониторингу, учету и контролю проектных работ на каждом из этапов ЖЦ программного продукта, в ходе которых достигаются цели проекта при нахождении компромисса между объемом работ, ресурсами, временем, качеством и рисками.

Цели программного проекта следует определять в виде желаемого результата, достигаемого командой проекта при его успешной реализации.

Формулирование желаемого результата программного проекта должно отражать конкретные бизнес-цели с учетом *правила* «*железного треугольника*» (рис. 6.1) [13].

Рис. 6.1 – «Железный треугольник» ограничений

Пример

Ни один из углов треугольника не может быть изменен без изменения других: например, чтобы уменьшить время, потребуется увеличить бюджет и/или сократить содержание.

С учетом наличия рыночной конкуренции к трем основным характеристикам «железного треугольника» следует добавить четвертую – *приемлемое качество*, которое определяется в виде совокупности нефункциональных требований к ПП (рис. 6.2).

Рис. 6.2 – Возможные варианты «железного треугольника» ограничений проекта

В этом случае желаемый результат программного проекта можно сформулировать следующим образом: проект должен быть реализован в нормативные сроки без превышения планового бюджета с заданными заказчиком функциональными и нефункциональными требованиями.

Наличие большой неопределенности в достижении конечных целей проекта объективно требует проведения мероприятий по выявлению и оценке возможных рисков.

Под риском программного проекта будем понимать наступление события, которое может возникнуть в процессе реализации программного проекта и негативно повлиять на степень достижения целей проекта.

Оптимальный вариант реализации программного проекта определяется как *компромисс* между сформулированными выше характеристиками результатов проекта и состоит в нахождении баланса, приемлемого для всех участников проекта:

1) заказчиков, которым нужна определенная функциональность в конкретные сроки при имеющемся бюджете;
2) исполнителей, которые обладают бюджетом, достаточным для реализации проекта в заданные сроки с расчетным уровнем трудоемкости проекта и соответствующим уровнем качества.

Проект считается успешным, если он выполнен в срок в соответствии со спецификациями функциональных и нефункциональных требований в пределах запланированного бюджета.

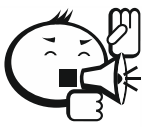
6.2 Этапы жизненного цикла программного проекта

Управление процессами разработки уникальных конечных продуктов и/или услуг происходит в рамках проектной деятельности организации.

В качестве документов, регламентирующих проектную деятельность, могут быть использованы:

1) Руководство к своду знаний по управлению проектами (Project Management Body of Knowledge – PMBOK);
2) ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств».

В стандарте PMBOK описаны пять групп управленческих процессов, соответствующих этапам жизненного цикла проекта, и десять областей знаний, которые используются менеджером проекта при определении содержания соответствующих этапов жизненного цикла проекта. Содержание видов деятельности, необходимых для управления проектом, изложено в описании 47 процессов.



Все пять групп управленческих процессов присущи любому проекту и должны выполняться в одной и той же последовательности: *инициация, планирование, исполнение, мониторинг и управление, завершение* [14].

В рамках процессов *инициации* определяются:

- бизнес-потребности потенциальных заказчиков, которым должен удовлетворять проект;
- причины, по которым данный проект является лучшей альтернативой для удовлетворения потребностей;
- внутренние и внешние заинтересованные в результатах проекта стороны;
- цели и содержание проекта;
- команда исполнителей проекта;
- финансовые ресурсы.

По завершении этапа разрабатывается концепция или устав проекта.

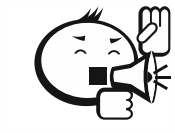
Группа процессов *планирования* состоит из процессов, определяющих содержание и последовательность работ проекта, необходимых и достаточных для достижения целей проекта.

В процессах планирования определяются состав и содержание работ проекта и их взаимосвязи; оцениваются трудоемкость каждой работы, типы и количество необходимых трудовых ресурсов; разрабатываются календарный план проекта, план распределения ресурсов, бюджетный план, план управления рисками.

В составе группы процессов *исполнения* рассматриваются вопросы организации работ по проекту: распределение обязанностей и ответственности; координация работы исполнителей и использования ресурсов; определение механизмов морального и материального стимулирования работы исполнителей.

Группа процессов *мониторинга и управления* обеспечивает: контроль, анализ и регулирование хода и эффективности выполнения как проекта в целом, так и его отдельных работ; выявление проблем, требующих внесения изменений в план, и инициацию внесения этих изменений; разработку рекомендаций по применению предупреждающих действий в отношении возможных проблем.

Группа процессов *завершения* обеспечивает: приемку-сдачу проекта заказчику; документирование и накопление знаний о положительных и отрицательных аспектах практического использования результатов проекта; проведение анализа эффективности практического использования результатов; внесение необходимых изменений в очередной версии проекта.



Описание десяти областей знаний состоит из следующих разделов:

1) *управление интеграцией проекта*: разработка плана проекта, организация исполнения плана проекта, контроль изменений в проекте;

2) *управление содержанием проекта*: формальное принятие решения о начале проекта, планирование объема работ, декомпозиция проекта на мелкие измеримые работы, определение трудозатрат, материальных и финансовых ресурсов по каждой работе;

3) *управление сроками проекта*: определение состава работ и их взаимосвязей, оценка длительностей работ, составление расписания проекта;

4*) управление стоимостью проекта*: планирование и оценка стоимости ресурсов, необходимых для работ; разработка бюджета, распределение затрат по работам проекта; контроль стоимости и управление изменениями бюджета;

5) *управление качеством проекта*: определение стандартов качества и средств для их достижения; регулярная оценка исполнения производственных процессов; мониторинг результатов проекта, определение их соответствия стандартам; выявление и устранение причин несоответствия качества;

6) *управление человеческими ресурсами*: документирование и назначение проектных ролей участникам проекта и структуры их отчетности; подбор и получение необходимых для проекта трудовых ресурсов; распределение персонала по работам проекта; формирование планов повышения квалификации исполнителей проекта; стимулирование индивидуальной и командной производительности труда;

7) *управление коммуникациями*: определение потребностей участников проекта в информации, планирование информационных

потоков; регулярное и своевременное обеспечение участников проекта необходимой информацией; сбор и распространение отчетности о текущем состоянии проекта; создание и утверждение отчетов, необходимых для формального завершения проекта или отдельных этапов проекта;

8) *управление рисками проекта*: разработка плана управления рисками, идентификация рисков и рискообразующих факторов, качественный и количественный анализ, планирование реагирования на риски, мониторинг и контроль процессов управления рисками.

9) *управление закупками проекта*: определение продуктов и услуг, которые необходимо привлечь извне для выполнения проекта; документирование требований к продуктам и услугам от внешних поставщиков; получение предложений, выбор поставщиков; регулирование отношений с поставщиками; подтверждение по выполнению условий контрактов; разрешение споров;

10) *управление заинтересованными сторонами проекта*: определение состава и ролевых функций участников проекта: инициатора, инвестора, заказчика, руководителя, куратора, соисполнителей; формирование регламентов по взаимодействию участников при реализации этапов жизненного цикла проекта.

Пример

Вариант распределения работ по этапам жизненного цикла, рекомендованных стандартом [7], приведен в таблице 6.1.

Таблица 6.1 – Распределение процессов управления проектом по областям знаний и этапам жизненного цикла

Область знаний	**Процессы управления проектами**				
	Инициация	**Планирование**	**Исполнение**	**Мониторинг и управление**	**Завершение**
1. Управление интеграцией проекта	1.1. Разработка устава проекта 1.2. Разработка предварительного описания содержания проекта	1.3. Разработка плана управления проектом	1.4. Руководство и управление исполнением проекта	1.5. Мониторинг и контроль работ проекта 1.6. Общее управление изменениями	1.7. Закрытие проекта

2. Управление содержанием проекта		2.1. Планирование содержания проекта 2.2. Определение содержания 2.3. Создание иерархической структуры работ (ИСР)		2.4. Подтверждение содержания 2.5. Управление содержанием	
3. Управление сроками проекта		3.1. Определение состава работ 3.2. Определение взаимосвязей 3.3. Оценка ресурсов работ 3.4. Оценка длительности 3.5. Разработка расписания		3.6. Управление расписанием	
4. Управление стоимостью проекта		4.1. Стоимостная оценка 4.2. Разработка бюджета расходов		4.3. Управление стоимостью	
5. Управление качеством проекта		5.1. Планирование качества	5.2. Процесс обеспечения качества	5.3. Процесс контроля качества	
6. Управление человеческими ресурсами проекта		6.1. Планирование человеческих ресурсов	6.2. Набор команды проекта 6.3. Развитие команды проекта	6.4. Управление командой проекта	
7. Управление коммуникациями проекта		7.1. Планирование коммуникаций	7.2. Распространение информации 7.3. Отчетность по исполнению	7.4. Управление участниками проекта	
8. Управление рисками проекта		8.1. Планирование управления рисками 8.2. Идентификация рисков 8.3. Качественный анализ рисков 8.4. Количественный анализ рисков 8.5. Планирование реагирования на риски		8.6. Мониторинг и управление рисками	

6.3 Управление содержанием и сроками реализации программного проекта

В общем случае управление содержанием и сроками реализации программного проекта описывается в виде взаимосвязанного комплекса работ, которые необходимо выполнить для достижения заданных целей проекта.

Ключевым элементом управления содержанием проекта является структурная декомпозиция работ проекта (Work Breakdown Structure, WBS). В состав работ должны быть включены все производственные, управленческие и административные действия, обеспечивающие разработку ПП и управление программным проектом.

В качестве исходных данных для проведения структурной декомпозиции работ можно использовать: модели жизненного цикла разработки ПП и состава архитектурного дизайна программного продукта.

Пример

Вариант распределения работ по элементам архитектурного дизайна программного продукта при использовании каскадной модели ЖЦ представлен в таблице 6.2 [7].

Таблица 6.2 – Дорожная карта распределения работ по объектам планирования

Работы	**Объекты планирования**						
	ПП	**Программный комплекс**		**Программа**		**Программный модуль**	
		1	2	1	2	1	2
1. Разработка функциональных требований к ПП	+						
2. Разработка системных требований к ПП	+						
3. Разработка ТЗ на ПП							
4. Проектирование архитектурного дизайна как многоуровневой структуры ПП	+						
5. Проектирование архитектуры программных комплексов		+	+				
6. Программирование программных модулей						+	+
7. Разработка технической документации на модуль						+	+
8. Модульное тестирование						+	+
9. Сборка программы				+	+		
10. Разработка пользовательской документации на программу				+	+		
11. Тестирование программ				+	+		
12. Интеграция программ в программный комплекс		+	+				
13. Разработка эксплуатационной документации на программный комплекс		+	+				
14. Тестирование программного комплекса		+	+				

15. Сборка ПП	+						
16. Разработка эксплуатационной документации на ПП	+						
17. Организация приемки-сдачи ПП	+						
18. Ввод в эксплуатацию ПП	+						

. . . . . .

Структурная декомпозиция работ может выполняться с разной степенью детализации. Если имеется некоторая неопределенность относительно длительности выполнения отдельных работ программного проекта, то работы отдаленной перспективы целесообразно планировать укрупненно, а ближайшего будущего – детально. Например, на поздней стадии проекта в качестве единицы планирования можно рассматривать работу «Тестирование программных комплексов», а далее, по мере выполнения проекта детализировать ее на две отдельные элементарные работы: тестирование программного комплекса 1 и тестирование программного комплекса 2. Такой метод планирования получил название «метод набегающей волны».

В этом случае структурная декомпозиция работ должна производиться до получения работ, которые понятны исполнителю и могут быть достаточно адекватно оценены по срокам исполнения и требуемым ресурсам. При этом продолжительность каждой работы на данном уровне декомпозиции не должна быть больше периода контроля выполнения работ проекта.

Структура зависимостей между работами и задачами проекта может быть представлена в виде иерархического вложенного списка задач (рис. 6.3).

Сформированный состав работ проекта является основой для разработки календарных планов реализации проекта, управления сроками выполнения отдельных работ. Каждая из работ обычно имеет ожидаемую (плановую) продолжительность (длительность), предполагаемую (ожидаемую) стоимость и требуемые ресурсы. Согласно стандарту PMBOK процессы управления сроками реализации проекта предназначены для составления базового календарного плана (расписания) проекта без учета ограничений на трудовые ресурсы.

Основой для разработки календарных планов являются сетевые модели (графики), описывающие логические зависимости между работами. Существует два способа графического представления сетевой модели: на «языке событий» и на «языке работ». Описание сетевой модели на «языке событий» предполагает, что каждая работа отображается на графике в виде «дуги», соединяющей два

кружка (события): событие, фиксирующее факт начала работы, и событие, фиксирующее факт окончания работы. Оба события имеют нулевую длительность. Этот способ построения сетевой модели описан в стандарте PMBOK.

1 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА
1.1 Разработка функциональных требований к ПП
……..
1.4 Проектирование архитектурного дизайна ПП
1.4.1 РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА 1
1.4.1.1 Проектирование архитектуры программного комплекса 1
1.4.1.1.1 Разработка программы 1
1.4.1.1.1.1 Проектирование архитектуры программы 1
1.4.1.1.1.1.1 Разработка программного модуля 1
1.4.1.1.1.1.1.1 Программирование модуля 1
1.4.1.1.1.1.1.2 Разработка документации на модуль 1
1.4.1.1.1.1.1.3 Тестирование модуля 1
1.4.1.1.1.1.2 Разработка программного модуля 2
1.4.1.1.1.1.2.1 Программирование модуля 2
…….
1.4.1.1.1.2 Сборка программы 1
……..
1.4.2 РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА 2
1.4.2.1 Проектирование архитектуры программного комплекса 2
1.4.2.1.1 Разработка программы 1
1.4.2.1.1.1 Проектирование архитектуры программы 2
……..

Рис. 6.3 – Иерархический список работ программного проекта

В сетевой модели на «языке работ» каждый кружок описывает конкретную работу, а дуги определяют логическую последовательность выполнения работ (рис. 6.4). В верхней части кружка отображается номер работы, в нижней – указывается ее длительность.

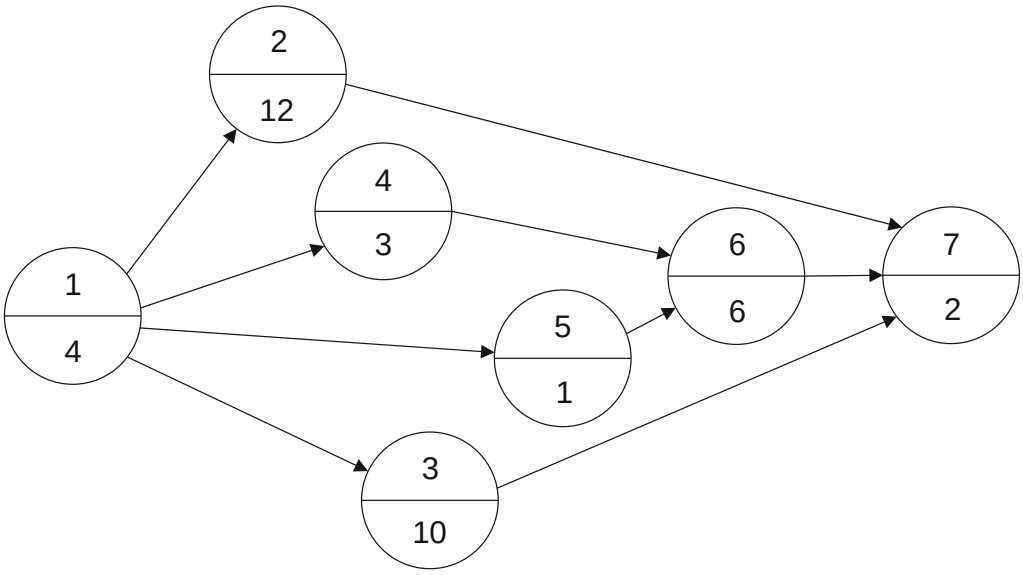
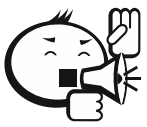


Рис. 6.4 – Пример сетевого графика на «языке работ»



При расчете базового календарного плана проекта определяются: ранняя дата начала, ранняя дата окончания, поздняя дата начала, поздняя дата окончания, полный резерв времени, критический путь.

Ранней датой начала выполнения работы называется наиболее раннее время его начала, не противоречащее взаимосвязям между работами и длительности их выполнения. Соответственно ранняя дата окончания выполнения работы отличается от ранней даты начала на величину длительности работы.

С точки зрения расчета времени раннего начала и окончания работ сетевой модели существуют два типа логических зависимостей между ними:

1. Взаимосвязь «финиш – старт», определяемая следующим правилом: предшествующая работа должна заканчиваться до того, как последующая работа может начаться. Такие две работы должны выполняться последовательно друг за другом, при этом возможно наличие некоторого времени задержки начала выполнения последующей работы: последующая работа должна начинаться не ранее чем через определенный интервал времени после окончания предшествующей работы.
2. Взаимосвязь «финиш – финиш», которая определяется следующим образом: предшествующая работа должна заканчиваться не ранее чем за t единиц времени до окончания последующей работы. Этот вид взаимосвязи предполагает наличие параллельно-последовательного порядка выполнения работ. При этом опережение начала выполнения последующей работы в зависимости от длительности этих работ определяется вычитанием от момента начала либо окончания последующей работы определенного количества периодов времени (t единиц времени опережения).

Поздняя дата начала выполнения работы – это самое позднее время его начала, при котором сохраняется общая длительность выполнения проекта и выполняются условия взаимосвязей между работами. Соответственно, поздняя дата окончания работы также отличается от ранней на величину длительности.

Полный резерв времени – это максимально допустимое время, на которое можно отложить момент окончания выполнения работы (сдвинуть время ее начала), при этом общая длительность выполнения проекта остается неизменной. Величина полного резерва времени вычисляется как разность между поздним и ранним временем начала выполнения работы.

Критический путь образуют работы, имеющие нулевой резерв времени. Работы с ненулевым резервом времени можно отложить на некоторый период времени, и наоборот, работы с нулевым резервом времени требуют особого внимания, поскольку их невыполнение в срок приводит к срыву выполнения проекта в целом.

6.4 Управление качеством программного проекта

Оценку качества программного проекта следует рассматривать с двух точек зрения:

- обеспечение качества выполнения функциональных и нефункциональных требований создаваемого программного продукта;
- обеспечение качества организации процессов управления программными проектами.

В первом случае для оценки качества можно использовать ГОСТ Р ИСО/МЭК 25040–2014 «Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Процесс оценки».

В этом документе для оценки качества ПП выделяются шесть характеристик: функциональность, надежность, удобство использования, эффективность, пригодность к обслуживанию, мобильность.

Функциональность – набор атрибутов, относящихся к оценке реализуемых в программном продукте функций и их конкретным свойствам (установленные или предполагаемые потребности):

- *пригодность* – атрибут, относящийся к наличию и соответствию набора функций конкретных задач;
- *правильность* – атрибуты, относящиеся к обеспечению правильности или соответствия результатов, или эффектов;
- *способность к взаимодействию* – атрибуты, относящиеся к способности ПП взаимодействовать с внешними системами;

- *согласованность* – атрибуты, обязывающие разработчиков придерживаться соответствующих стандартов или соглашений, или положений законов, или подобных рекомендаций;
- *защищенность* – атрибуты, относящиеся к способности предотвращать несанкционированный доступ, случайный или преднамеренный, к программам и данным.

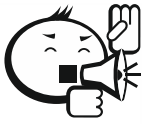
Надежность – набор атрибутов, относящихся к способности программного продукта сохранять качества функционирования при установленных условиях за установленный период времени:

- *стабильность* – атрибуты, относящиеся к частоте отказов при ошибках в программе;
- *устойчивость к ошибке* – атрибуты, относящиеся к способности поддерживать определенный уровень качества функционирования в случаях программных ошибок или нарушения определенного интерфейса;
- *восстанавливаемость* – атрибуты, относящиеся к возможности восстанавливать уровень качества функционирования и данные, непосредственно поврежденные в случае отказа, а также к времени и усилиям, необходимым для этого.

Эффективность – набор атрибутов, относящихся к соотношению между уровнем качества функционирования программного продукта и объемом используемых ресурсов при установленных условиях:

- *временная экономичность* – атрибуты, относящиеся к временам отклика и обработки и к скоростям выполнения функций;
- *ресурсная экономичность* – атрибуты, относящиеся к объему используемых ресурсов и продолжительности использования при выполнении функции.

Оценку качества организации процессов управления программными проектами можно проводить на основе международного стандарта CMM (Capability Maturity Model for Software) – модель зрелости процессов разработки программного обеспечения [13].



В стандарте определено пять уровней технологической зрелости, по которым компания-разработчик может оценивать и совершенствовать процессы управления программными проектами: начальный, повторяемый, определенный, управляемый, оптимизируемый.

Каждый уровень технологической зрелости соответствует определенному этапу развития компании по управлению и непрерывному совершенствованию процесса разработки ПП:

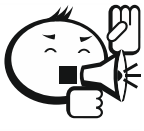
1. Начальный. Технология управления разработкой ПП характеризуется как произвольная, пригодная только для некоторых случаев, скорее даже хаотическая. Лишь некоторые процессы ЖЦ определены, успех проекта в основном зависит от компетенции отдельных сотрудников.
2. Повторяемый. Базовые процессы управления программным проектом определены и позволяют отслеживать затраты, график работы и функциональности программного продукта. Соблюдается необходимый порядок выполнения процессов и обеспечивается возможность повторения достижений, полученных ранее при выполнении подобных проектов.
3. Определенный. Базовые, вспомогательные и организационные процессы документированы, стандартизованы и интегрированы в унифицированную для данной компании технологию управления программными проектами, которая и используется при разработке ПП.
4. Управляемый. Управление всеми процессами по разработке ПП осуществляется по количественным оценкам. Детальные и объективные показатели о качестве исполнения процессов разработки ПП и характеристики самого продукта в соответствии с регламентом процессов разработки собираются и накапливаются.
5. Оптимизируемый. Совершенствование процессов разработки ПП осуществляется непрерывно, на основе количественного анализа эффективности процессов и использования инновационных методов и технологий.

Модель СММ не содержит никаких численных критериев оценки, рекомендаций и не указывает, как конкретно оценить продукт, а лишь показывает, что надо сделать для достижения требуемого качества программного обеспечения. СММ также содержит способы контроля за правильностью выполнения ключевых действий и методы их корректировки.

6.5 Управление рисками программного проекта

6.5.1 Риски и рискообразующие факторы программного проекта

Под риском проекта понимается событие или условие, которые могут произойти либо не произойти в будущем при реализации программного проекта и негативно повлиять на степень достижения одной или нескольких характеристик целей проекта.



Учитывая явную логическую взаимосвязь между целями проекта и возможными рисками, можно предположить, что при разработке программного проекта могут возникнуть четыре типа (категории) рисков [7]:

- срыв плановых сроков проекта;
- превышение стоимости (бюджета) проекта;
- критические отклонения по составу и содержанию проекта (невыполнение функциональных требований);
- критическое отклонение по показателям качества проекта (невыполнение нефункциональных требований).

Появление каждого из рисков возможно при наличии причин (процессов или явлений), способствующих его возникновению и поясняющих, почему наступление риска неизбежно. Такие явления принято называть *рискообразующими факторами.*

Для стандартизации и унификации терминологии рискообразующих факторов, оценки их влияния на конкретные цели и фазы программного проекта необходима систематизация и классификация факторов по определенным признакам. В данном случае для классификации факторов риска используется иерархический метод классификации, при котором множество рискообразующих факторов последовательно, в соответствии с выбранными основаниями (признаками) классификации, разбивается на подмножества (рис. 6.5) [7].

Рис. 6.5 – Классификация факторов риска программного проекта

На первом уровне классификатора в качестве основания классификации используется модель жизненного цикла программного проекта: инициация – разработка – продвижение – внедрение.

На втором уровне для каждого этапа жизненного цикла ПП можно выделить внешние и внутренние факторы. *Внешние факторы* – это события, которые лежат за пределами контроля и влияния команды проекта. *Внутренние факторы* (специфичные для конкретной компании) определяют способность самой организации успешно реализовать проект.

На третьем уровне проявление внешних факторов обусловливается как политикой государства в отношении бизнеса малых IT-компаний, так и различными ситуациями на продуктовом, финансовом рынках и рынке труда. Классификатор внутренних факторов определяется составом системной модели деятельности: средствами деятельности, предметами деятельности, кадрами, технологией.

Четвертый уровень представляет собой набор первичных факторов риска. При этом допускается возможность принадлежности одного и того же фактора разным основаниям классификации.

В соответствии с предложенной классификацией ниже приводятся примеры внешних и внутренних факторов риска при выводе на рынок ПП [7]. С точки зрения маркетингового подхода конечную цель можно определить как «достижение определенного объема продаж ПП в определенном интервале времени при ограничениях на бюджет рекламной компании (программы)». Учитывая в приведенных определениях явную логическую взаимосвязь между целями и возможными рисками, можно предположить, что в неблагоприятной рыночной ситуации возможен «срыв плановых показателей по объему продаж».

Внутренние факторы риска предлагается классифицировать по продукту, персоналу и технологии реализации продукта.

Продукт: нереальные сроки выхода на планируемые объемы продаж; ошибки в расчетах трудоемкости и финансовых затрат на разработку и продвижение программного продукта; появление «забытых» работ.

Персонал: отсутствие опыта, необходимого для разработки и реализации программы; саботаж отдельных членов команды; недостатки во внутренней организации работ; неумение работать в реальном времени.

Технологии реализации продукта: ошибки при выборе программно-аппаратной платформы; ошибки выбора каналов и инструментов продвижения; отсутствие эффективного взаимодействия с потенциальными пользователями.

Внешние факторы риска: государство, финансовый рынок, рынок труда, продуктовый рынок (потребители), продуктовый рынок (конкуренты).

Государство: изменение нормативно-правовых механизмов ведения бизнеса в ИТ-отрасли; отсутствие устоявшейся законотворческой практики по защите авторских и имущественных прав ПП.

Финансовый рынок: колебания курса валют; изменение ставок по кредитам.

Рынок труда: отсутствие специалистов требуемой квалификации.

Продуктовый рынок (*потребители*): несоответствие функциональных характеристик ПП потребностям потребителей; несовместимость предлагаемого продукта с программно-аппаратной средой потребителей; несоответствие рыночной цены ПП возможностям потенциальных потребителей; низкий уровень подготовки пользователей у потенциальных потребителей ПП.

Продуктовый рынок (*партнеры, конкуренты*): появление на рынке новых аналогичных продуктов; непредсказуемое поведение конкурентов; пиратское распространение копий ПП.

Задачами менеджмента по управлению рисами программного проекта являются: своевременное выявление, описание, оценивание рискообразующих факторов; выработка рекомендаций по реагированию на выявленные рискообразующие факторы.

6.5.2 Качественный и количественный анализ рискообразующих факторов

Идентификация – этап, позволяющий выявить и коллективно обсудить возможность проявления риска и рискообразующих факторов, способных повлиять на цели проекта, документально описать результаты в виде логически увязанных характеристик.

Последовательность действий команды по выявлению и описанию рисков может быть определена на основе предложенного классификатора рискообразующих факторов, а в качестве методов и инструментов могут использоваться метод мозгового штурма, опросы экспертов, SWOT-анализ.

Описание факторов риска следует проводить на естественном языке по схеме: условия возникновения, последствия от проявления, влияние на цели проекта. Условие содержит описание причины, которая может сделать результаты реализации проекта убыточными. Последствие описывает нежелательную ситуацию при наступлении рискоообразующего фактора, которую следует избегать. Воздействие на цели отражают негативные изменения характеристик целей программы.

Пример

Пример идентификации риска при выводе ПП на рынок с описанием факторов представлен в таблице 6.3.

Таблица 6.3 – Фрагмент описания схемы рискообразующих факторов

Факторы	**Описание фактора**		
	Условие	**Последствия**	**Воздействие на цели программы продвижения**
…	…	…	…
2. Изменение экономической ситуации при выводе ПП на рынок	Экономический кризис	Изменение платежеспособности потребителей	Сокращение объемов продаж
3. Появление новых аналогичных продуктов	Выход на рынок новых аналогичных продуктов	Усиление конкуренции	Сокращение объемов продаж
4. Ошибки выбора каналов и инструментов коммуникаций	Снижение необходимого уровня информирования целевой аудитории	Несоответствие плановых и фактических показателей результативности программы продвижения	Сокращение объемов продаж

На этапе анализа необходимо определять следующие качественные и количественные оценки рисков и рискообразующих факторов:

- *вероятность появления рискообразующих факторов и уровень* (*степень*) *их негативного влияния на цели проекта;*
- *временной диапазон проявления рискообразующих факторов;*
- *множество рискообразующих факторов, оказывающих критическое влияние на результаты проекта и требующих скорейшего реагирования.*

Значения показателей вероятности и уровня негативного воздействия могут оцениваться как в количественных, так и в качественных шкалах. Среди количественных методов оценки вероятности рискообразующих факторов и их влияния на цели проекта наиболее часто используется метод PERT-анализа (Project Evaluation and Review Technique) [13]. Суть его заключается в том, что для каждой характеристики эксперту необходимо указывать три оценки – оптимистическую, наиболее вероятную (реалистическую) и пессимистическую.

Тогда вероятность наступления рискообразующих факторов можно вычислять по следующей формуле:

$$P(x_j) = \left| p_1(x_j) + 4p_2(x_j) + p_3(x_j) \right| / 6,$$

где $p_1(x_j)$, $p_2(x_j)$, $p_3(x_j)$ – соответственно оптимистическая, пессимистическая и реалистическая вероятности наступления фактора.

Качественная оценка рискообразующих факторов и их влияния на цели проекта проводится в шкале интервалов с несколькими градациями (табл 6.4). Ввиду присутствия неопределенности интервалы оценки могут пересекаться.

Таблица 6.4 – Шкала оценивания вероятности проявления и силы воздействия рискообразующих факторов

Вероятность	**Очень низкая**	**Низкая**	**Умеренная**	**Высокая**	**Очень высокая**
Интервал	Менее 0,15	[0,1; 0,4]	[0,2; 0,6]	[0.5; 0,9]	Более 0,8

По результатам идентификации и анализа описанных выше показателей вычисляется интегральная оценка критичности рискообразующих факторов как произведение величин <вероятность – сила воздействия>.

Следующей важной характеристикой рисков и рискообразующих факторов является временной диапазон проявления (*близость их наступления*). Естественно, что при прочих равных условиях рискам, которые могут осуществиться уже завтра, следует сегодня уделять больше внимания, чем тем, которые могут произойти не ранее чем через полгода. Возможная шкала оценки близости представлена в таблице 6.5.

Ранжирование множества рискообразующих факторов, оказывающих критическое влияние на результаты проекта и требующих соответствующего управления, проводится на основе вычисления рейтинга.

Таблица 6.5 – Шкала оценивания близости наступления рисков и рискообразующих факторов

Количественное значение близости наступления	Больше чем через …	От … до …	Меньше чем через …
Качественное значение близости наступления	Очень нескоро	Не очень скоро	Очень скоро

Возможные правила, позволяющие определить вычисления рейтинга рискообразующих факторов, представлены в таблице 6.6 [11].

Таблица 6.6 – Оценка рейтинга рискообразующих факторов

Близость наступления	**Степень критичности**				
	Невысокая	**Умеренная**	**Средняя**	**Высокая**	**Катастрофическая**
Очень нескоро	Низкий	Низкий	Средний	Высокий	Высокий
Не очень скоро	Низкий	Средний	Средний	Высокий	Высокий
Очень скоро	Низкий	Средний	Высокий	Очень высокий	Очень высокий

Очень высокий рейтинг (4) присваивается факторам, требующим немедленного реагирования, соответственно, самый низкий рейтинг (1) присваивается рискообразующим факторам, критичность которых невысока и их наступление отложено на длительный срок.

Ранжирование факторов риска позволяет команде проекта распределить их по категориям опасности последствий проявления:

- рейтинг 4 – рискообразующие факторы, требующие немедленного реагирования;
- рейтинг 3 – рискообразующие факторы, реагирование на которые можно выполнить позже;
- рейтинг 2 – рискообразующие факторы, требующие дополнительного рассмотрения (включая количественный анализ);
- рейтинг 1 – рискообразующие факторы, за которыми в дальнейшем должно проводиться наблюдение.

6.5.3 Стратегии управления рисками

Процессы ***планирования мероприятий по реагированию на риски*** *предполагают выбор стратегии по снижению угроз и разработку планов мероприятий по реализации стратегий. Согласно* [14], *возможны четыре вида таких стратегий: уклонение от риска, передача риска, снижение риска, принятие риска.*

Уклонение от риска предполагает разработку комплекса мероприятий по нейтрализации критических рискообразующих факторов, т. е. изменение плана

управления проектом таким образом, чтобы исключить влияние негативных факторов на цели проекта или скорректировать целевые показатели, находящиеся под угрозой, например, отказаться от реализации рискованного функционального требования.

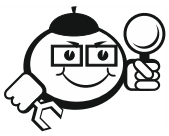
Передача риска подразумевает переложение негативных последствий от проявления рискообразующего фактора на третью сторону (но риск при этом остается), например, заказать разработку рискованного компонента «на стороне». Данная стратегия эффективна для нейтрализации критических рискообразующих факторов, влияющих на бюджет проекта. Условия передачи ответственности третьей стороне должны определяться в контракте (гарантии выполнения контракта, гарантийные обязательства).

Снижение риска предполагает понижение вероятности и/или последствий негативного проявления рискообразующего фактора до приемлемых пределов, например, увеличить сроки выполнения проекта, понизить значения ряда показателей качества ПП. Принятие предупредительных мер по снижению вероятности наступления фактора или его последствий зачастую оказывается более эффективным, нежели действия по устранению негативных последствий, предпринимаемые после наступления события.

Принятие риска предполагается в тех случаях, когда избежать проявления рискообразующих факторов маловероятно и команда проекта не нашла эффективных мероприятий реагирования на риски. Реализация данной стратегии возможна в двух вариантах: активном либо пассивном.

Пассивное принятие данной стратегии не предполагает проведения каких-либо предупредительных мероприятий, оставляя команде проекта право действовать по собственному усмотрению в случае наступления негативных событий. Наиболее распространенной формой активного принятия данной стратегии является создание резерва на непредвиденные обстоятельства в виде возможности привлечения дополнительных финансовых и/или трудовых ресурсов либо корректировки сроков реализации проекта.

По каждому из мероприятий первой группы назначают одного или нескольких ответственных лиц («ответственных за реагирование на риски»), определяются бюджет и сроки выполнения мероприятия. Прошедший экспертизу и утверждений план мероприятий должен быть включен в общий процесс управления изменениями программного проекта.



Пример

Ниже приведены примеры мероприятий, направленных на снижение ряда рискообразующих факторов. Так, например, негативное влияние фактора *<неполные или нечеткие требования к программному продукту>* можно понизить за счет:

- изменения стоимости и сроков реализации проекта при каждом добавлении или корректировке требований;
- согласования с заказчиком подробного перечня требований и включение этого списка в контракт на разработку ПП;
- использования моделей ЖЦ, позволяющих периодическое уточнение требований;
- введения буферных работ с соответствующими ресурсами и длительностью выполнения.

Риски, связанные с *изменениями ситуации на финансовом рынке*, можно снизить путем внесения в контракт условий, предусматривающих корректировку стоимости проекта в случае кризисных явлений, не зависящих от воли сторон (изменение курса валют). Потери, связанные с *ненадежной работой аутсорсинговых компаний*, можно избежать путем внесения в контракт пункта о штрафных санкциях за нарушение условий поставки продукта.

Мониторинг и управление рисками *представляет собой процесс идентификации, анализа рисков и рискообразующих факторов и планирования мероприятий по реагированию на новые риски, отслеживания ранее идентифицированных рисков, а также проверки и исполнения мероприятий по реагированию на риски и оценке эффективности их выполнения.*

При этом необходимо постоянно решать следующие задачи: пересмотр рисков, аудит рисков, анализ отклонений и трендов.

Пересмотр рисков предполагает регулярную, согласно принятым регламентам, идентификацию, анализ и планирование реагирования на новые риски. Управление рисками проекта должно быть одним из пунктов повестки дня всех совещаний команды проекта.

Аудит рисков предполагает изучение и предоставление в документальном виде результатов оценки эффективности выполнения мероприятий по реагированию на риски, требующие немедленного реагирования, изучение основных причин их возникновения.

На основании *анализа отклонений и трендов* проекта можно прогнозировать на очередной плановый период влияние негативных последствий проявления рискообразующих факторов на цели проекта. Контроль и анализ трендов может повлечь за собой выбор альтернативных стратегий, принятие корректив, перепланировку проекта для достижения базового плана.

Контрольные вопросы по главе 6

1. Дайте понятие программного проекта и перечислите его специфические особенности.
2. Приведите определение программного продукта. Перечислите свойства ПП как объекта интеллектуальной собственности.
3. Дайте понятия цели, результата и ограничений программного проекта. Перечислите и прокомментируйте требования к формулировке целей.
4. Раскройте смысл характеристик «железного треугольника» при управлении программными проектами. В чем состоит процедура достижения компромисса между характеристиками?
5. Приведите понятие жизненного цикла программного продукта и назовите стандарты, регламентирующие этапы ЖЦ.
6. Перечислите и прокомментируйте содержание девяти областей знаний стандарта PMBOK.
7. Перечислите и прокомментируйте содержание пяти этапов жизненного цикла программного проекта стандарта PMBOK.
8. Перечислите и прокомментируйте содержание процессов управления программным проектом стандарта ГОСТ Р ИСО/МЭК 12207–2010.
9. Дайте определение и приведите примеры понятий «риск» и «рискообразующий фактор».
10. Приведите пример и прокомментируйте по схеме «условие» – «последствие» – «воздействие» описание внутренних факторов риска программного проекта.

11. Приведите пример и прокомментируйте по схеме «условие» – «последствие» – «воздействие» описание внешних факторов риска программного проекта.
12. Раскройте содержание и методы описания показателей вероятности и негативных последствий рискообразующих факторов.
13. Поясните процедуру ранжирования рискообразующих факторов по степени опасности последствий от их наступления.
14. Раскройте содержание модели функциональных зависимостей определения рисков программного проекта.
15. Раскройте содержание стратегий по управлению рисками, приведите примеры конкретных мероприятий по каждой из стратегий.
16. Раскройте содержание этапа мониторинга и управления рисками.

Заключение

Содержание учебного пособия направлено на формирование у студента осознания социальной значимости будущей профессии, мотивации к получению профессиональных знаний, понимания основных концепций и содержания программной инженерии как методологии индустриального проектирования программных продуктов.

В этой связи в учебном пособии описана модель технологического процесса промышленного производства программного продукта, приводятся материалы:

- об отечественных и зарубежных стандартах, регламентирующих процессы жизненного цикла программных продуктов и кратное описание содержания этих процессов;
- о моделях разработки программных продуктов;
- о перечне и содержании этапов и областей знаний управления программными проектами, выводе на рынок и коммерциализации программных продуктов.

Содержание самостоятельной работы обучающихся при изучении данной дисциплины направлено на получение знаний и практических навыков: по поиску и обработке информации из различных электронных источников о направлениях программной инженерии как прикладной науке и оформлению полученных результатов в виде аналитического обзора по одной из тем, вынесенных для самостоятельной работы

Автор искренне надеется, что изучение дисциплины повысит у студентов мотивацию к получению знаний и подтвердит правильность выбранной ими профессии.

Литература

1. Федеральный государственный образовательный стандарт высшего образования по направлению подготовки 09.03.04 «Программная инженерия» (утвержден 19.09.2017, приказ № 920) [Электронный ресурс]. – Режим доступа: http://www.edu.ru/file/docs/2017/09/m920.pdf#page=3 (дата обращения: 15.05.2019).
2. Профессиональные стандарты: Связь, информационные и коммуникационные технологии [Электронный ресурс]. – Режим доступа: http://fgosvo.ru/docs/101/69/2/6 (дата обращения: 15.05.2019).
3. Этика профессиональной деятельности. Кодекс этики и профессиональной деятельности в области программной инженерии [Электронный ресурс]. – Режим доступа: https://club.shelek.ru/viewart.php?id=277 (дата обращения: 04.05.2019).
4. Ехлаков, Ю. П. Организация бизнеса на рынке программных продуктов : учебник [Электронный ресурс] / Ю. П. Ехлаков. – Томск : Изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2012. – 312 с. – Режим доступа: https://edu.tusur.ru/publications/970 (дата обращения: 04.05.2019).
5. Microsoft Solutions Framework. Модель процессов MSF. Версия 3.1 : пер с англ. [Электронный ресурс]. – Томск, 2003. – 41 с. – Режим доступа: https://tusur.ru/urls/vwa7eupd (дата обращения: 15.05.2019).
6. Моделирование и анализ бизнес-процессов : учеб. пособие [Электронный ресурс] / М. П. Силич, В. А. Силич. – Томск, 2011. – 213 с. – Режим доступа: https://edu.tusur.ru/publications/673 (дата обращения: 04.05.2018).
7. Управление программными проектами : учебник [Электронный ресурс] / Ю. П. Ехлаков. – Томск, 2015. – 217 с. – Режим доступа: https://edu.tusur.ru/publications/6024 (дата обращения: 04.05.2019).
8. Халл, Э. Разработка и управление требованиями. Практическое руководство пользователя [Электронный ресурс] / Элизабет Халл, Кен Джексон, Джереми Дик. – 2-е изд. – Режим доступа: http://www.interface.ru/iarticle/files/19771_42776753.pdf (дата обращения: 15.05.2019).

9. Вигерс, К. Разработка требований к программному обеспечению [Электронный ресурс] / Карл Вигерс, Джой Битти. – СПб. : БХВ-Петербург, 2013. – Режим доступа: https://avidreaders.ru/book/razrabotka-trebovaniy-k-programmnomu-obespecheniyu.html (дата обращения: 15.05.2019).
10. Руководство к своду знаний по программной инженерии. The Guide to the Software Engineering Body of Knowledge, SWEBOK, IEEE Computer Society Professional Practices Committee, 2004 [Электронный ресурс]. – Режим доступа: https://b-ok.cc/ireader/3109458 (дата обращения: 15.05.2019).
11. Бек, К. Экстремальное программирование: разработка через тестирование : пер. с англ. / Кент Бек ; пер. П. Анджан. – СПб. : Питер, 2003. – 224 с.
12. Ехлаков, Ю. П. Модели и алгоритмы поддержки принятия решений при продвижении на промышленные рынки прикладных программных продуктов : монография / Ю. П. Ехлаков, Д. Н. Бараксанов, Н. В. Пермякова. – Томск : Изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2015. – 132 с.
13. Фатрелл, Роберт Т. Управление программными проектами. Достижение оптимального качества при минимуме затрат / Роберт Т. Фатрелл, Дональд Ф. Шафер, Линда И. Шафер. – М. : ИД «Вильямс», 2004. – 1136 с.
14. Руководство к своду знаний по управлению проектами (PMBOK). – 4-е изд. – М. : Project Management Institute, 2010. – 496 с.

Глоссарий

Анализ требований – отображения функций системы и ее ограничений в модели предметной области.

Артефакт – любой продукт деятельности специалистов по разработке ПО.

Архитектура программной системы – определение системы в терминах подсистем и интерфейсов между ними, отображающая правила декомпозиции проблемы.

Бизнес-процесс – множество внутренних упорядоченных видов деятельности, организация по преобразованию исходных ресурсов в готовую продукцию (услугу).

Водопадная (каскадная) модель – схема работ, в которой каждая из работ выполняется один раз и в том порядке, который указан в модели жизненного цикла.

Гарантия качества программного обеспечения – действия на каждом этапе жизненного цикла по проверке и подтверждению достигаемого качества соответственно стандартам и процедурам.

Диаграмма – графическое представление моделирования системы с помощью классов, сценариев, состояний и т. п.

Динамическое тестирование – выполнение программ для обнаружения ошибок, установления их причины и устранения.

Жизненный цикл системы (*ЖЦ*) – непрерывный процесс, который начинается с момента принятия решения о необходимости ее создания и заканчивается в момент ее полного изъятия из эксплуатации.

Иерархия – упорядочение абстракций, расположение их по уровням.

Инженерия – применение научных результатов и дисциплины управления программированием задач в целях получения пользы от свойств продуктов, способов взаимосвязи и выполнения.

Инженерия качества – процесс управления предоставлением продуктам программного обеспечения свойств качества (надежность, сопровождаемость и т. п.).

Инженерия требований – сбор, анализ, оформление условий и ограничений на разработку системы в виде спецификации, согласованной как заказчиком, так и исполнителем.

Качество программного обеспечения – совокупность свойств, которая определяет пригодность программного обеспечения удовлетворить заказчика в соответствии с его требованиями к разработке.

Компонент – тип, класс, проектное решение, документация или иной продукт программной инженерии, приспособленный для практического использования.

Компонентная разработка – конструирование программного обеспечения путем композиции готовых компонентов, сохраняемых в каталогах.

Конечные пользователи системы – профессиональные лица, для потребностей которых заказывается компьютерная система.

Конфигурация – вариант (версия) архитектуры изготовленной программной системы из отдельных экземпляров компонентов и подсистем.

Критерий – количественная или качественная характеристика состояния системы, позволяющая оценить степень достижения цели и сформулировать решающие правила выбора средств (способов, технологий) достижения цели.

Модель жизненного цикла – типичная схема последовательности работ на процессах разработки программного продукта.

Модель процессов – определенная последовательность действий, сопровождающая изменение состояния программного объектов.

Модульность – свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули.

Надежность программной системы – способность системы сохранять свои свойства (безотказность, устойчивость и др.) в процессе преобразования исходных данных в результаты в течение определенного промежутка времени при определенных условиях эксплуатации.

Нефункциональные требования – требования, которое характеризуют организационные, исполнительские, операционные аспекты работы программной системы в среде реализации.

Отладка – проверка программы на наличие в ней ошибок и их устранение без внесения новых.

Объектно-ориентированная модель – структура из совокупности объектов, которые взаимодействуют между собою, обладают свойствами и поведением.

Оценивание качества – действия, направленные на определение степени удовлетворения программного обеспечения требованиям, соответствующим его предназначению.

Пакет – программная структура с общим механизмом организации элементов (объектов, классов) в группы, начиная от системы (стереотип «система») и к ее подсистемам различного уровня детализации.

Переносимость системы – возможность изменять сервис системы (ОС, связи, сетевые коммуникации, данные СУБД и т. п.) путем настройки модулей на новые условия среды или платформы.

План тестирования – описание стратегии, ресурсов и график тестирования отдельных компонентов и системы в целом.

Повторное использование – использование в качестве готовой порции любых формализованных знаний, полученных при реализации программных систем.

Повторно используемый компонент (ПИК) – фрагмент знаний о минувшем опыте программирования системы, представленный так, чтобы его можно использовать не только его разработчиками, но и пользователями после соответствующей адаптации к новой среде.

Предметная область представляет собой набор бизнес-процессов организации, адекватно описывающих деятельность организации по удовлетворению потребности общества в определенных продуктах и услугах.

Прикладная система – продукт программной инженерии, предназначенный для выполнения конкретных задач конечного пользователя.

Программная инженерия – система методов, средств и дисциплины планирования, разработки, эксплуатации и сопровождения программного обеспечения, способного к массовому воспроизводству.

Процесс разработки – действия разработчика по инженерии требований, проектированию, кодированию и тестированию программного продукта.

Процесс сдачи – действия по передаче разработанного продукта покупателю.

Процесс эксплуатации – действия по обслуживанию системы пользователем.

Процесс сопровождения – действия по управлению модификациями и поддержкой системы в актуальном состоянии при выполнении функций системы или изъятии системы из употребления.

Проектирование – преобразование требований в последовательность проектных решений и их в архитектуру из программных компонентов.

Проектирование концептуальное – уточнение понимания и согласование деталей требований к системе.

Проектирование архитектурное – определение структурных особенностей строящейся системы.

Проектирование техническое – отображение требований среды функционирования и разработки системы путем определения всех конструктивных элементов и их композиций.

Проектирование детальное – определение подробностей реализации функций для заданной среды и связей между соответствующими компонентами системы.

Реализация программной системы – преобразования проектных решений в работающую систему (синонимы: кодирование, конструирование).

Сертификация программного продукта – процесс для установления соответствия программной продукции (процесса или услуг) конкретному стандарту или техническим условиям со специальным знаком или свидетельством.

Спецификация – описание алгоритма, правил, ограничений действий объектов с учетом стандартов, критериев качества и др.

Средства проектирования – создание моделей программного продукта на основе моделей предметной области (например, IBM Rational Rose, Sybase Power Designer).

Статическое тестирование – анализ и рассмотрение спецификаций компонентов на правильность представления без их выполнения на компьютере.

Сопровождение – работы по внесению изменений в программную систему после того, как она передана пользователю для эксплуатации.

Структура системы – множество элементов и отношений между ними.

Сценарий – конкретная последовательность действий, которая иллюстрирует поведение и выполнение экземпляра прецедента.

Тест – некоторая программа, предназначенная для проверки правильности ее работы и выявления в ней ошибочных ситуаций.

Тестовые данные – данные, которые готовятся на основе документов программы или спецификаций для проверки работы программной системы.

Тестирование – способ семантической отладки (проверки) программы, который состоит в выполнении последовательности различных контрольных наборов тестов и сверки с известным результатом.

Технология разработки программного обеспечения – это упорядоченная совокупность взаимосвязанных этапов создания программного продукта и набор инструментальных средств их реализации.

Требование – соглашение или договор между заказчиком и исполнителем системы относительно ее работы.

Унаследованная система – существующая действующая система, созданная любыми методами и технологиями для поддержки некоторых процессов бизнеса.

Управление качеством – комплекс способов и системной деятельности по планированию, управлению и оценке качества программного обеспечения.

Функция – содержание действий, выполнение которых возлагается на элемент системы при заданных требованиях, условиях и ограничениях.

Функциональные требования – требования, которые определяют цели и функции системы и принципы их выполнения на компьютере.

Функциональная полнота – атрибут, показывающий степень достаточности основных функций для решения специальных задач соответственно назначению программного обеспечения.

Функциональная структура – структура, элементами которой являются функции, реализуемые подразделениями предприятия, а отношениями – связи, обеспечивающие передачу предметов труда.

Характеристики качества – функциональность, надежность, удобство, эффективность, сопровождаемость, переносимость и тому подобное.

Эксплуатация – действия по выполнению готовой программной системы.

UML (Unified Modeling Language) – диаграммный способ (язык) для спецификации, визуализации, конструирования и документирования продуктов на процессах ЖЦ.

Учебное издание

Юрий Поликарпович Ехлаков

ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Учебное пособие

Корректор А. Н. Миронова
Оригинал-макет Г. Д. Дурягиной

Подписано в печать 16.09.2019. Формат 60x84 1/16.
Бумага офсетная. Гарнитура Times.
Усл. печ. л. 7,44.
Тираж 150 экз. Заказ № .

Издательство «Эль Контент»
634061, г. Томск, ул. Киевская, д. 57, оф. 27

ISBN 978-5-4332-0280-1