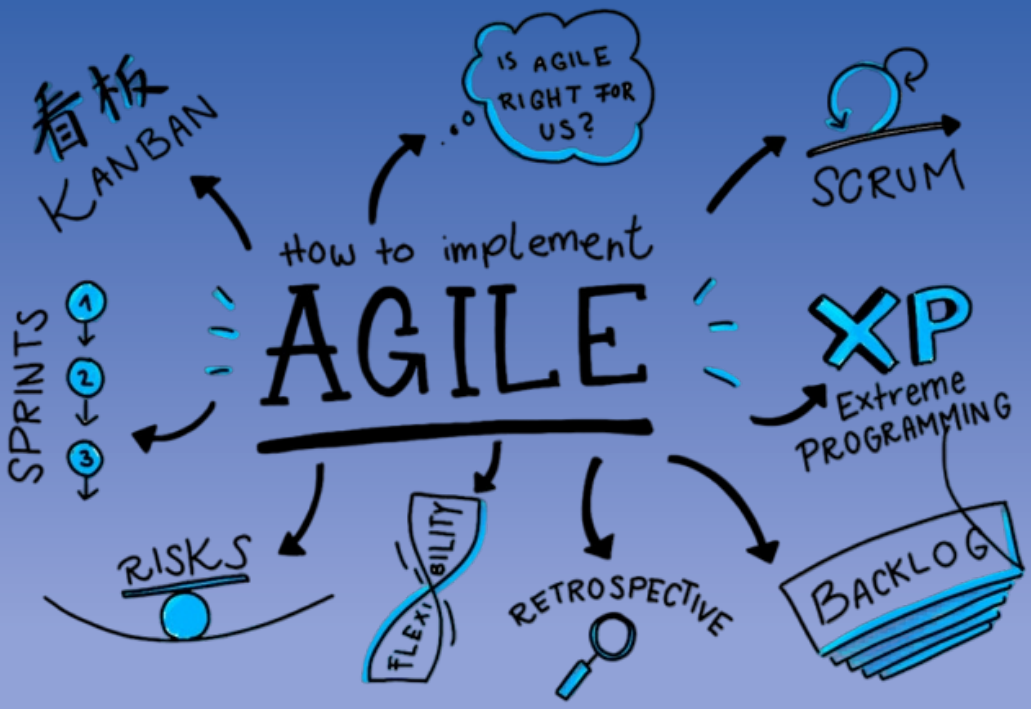
Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

Д. Б. БЕРГ
О. М. ЗВЕРЕВА
А. Ю. ВИШНЯКОВА

УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие



Министерство науки и высшего образования
Российской Федерации

Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

Д. Б. Берг, О. М. Зверева, А. Ю. Вишнякова

Управление жизненным циклом информационных систем

Учебное пособие

Рекомендовано методическим советом
Уральского федерального университета
для студентов вуза, обучающихся
по направлениям подготовки
09.03.01 — Информатика и вычислительная техника,
09.03.03, 09.04.03 — Прикладная информатика,
09.03.04 — Программная инженерия

Екатеринбург
Издательство Уральского университета
2022

УДК 004.00(075.8)
ББК 32.97я73
Б48

Рецензенты:
кафедра естественно-научных дисциплин УрГУПС (завкафедрой, д-р физ.-мат. наук, проф. *Г. А. Тимофеева*);
старший научный сотрудник Института математики и механики им. Н. Н. Красовского УрО РАН, канд. физ.-мат. наук *Д. Г. Ермаков*

Научный редактор — канд. физ.-мат. наук, доц. *А. С. Кощеев*

На обложке — изображение с сайта https://clck.ru/32RBAJ.

Берг, Дмитрий Борисович.
Б48 Управление жизненным циклом информационных систем : учебное пособие / Д. Б. Берг, О. М. Зверева, А. Ю. Вишнякова ; М-во науки и высшего образования РФ. — Екатеринбург : Изд-во Урал. ун-та, 2022. — 94, [2] с.

ISBN 978-5-7996-3594-7

В учебном пособии рассмотрены основные понятия, модели и методологии жизненного цикла разработки информационных систем.

Основная задача пособия — развитие навыков управления жизненным циклом информационных систем у студентов и подготовка их к решению практических задач разработки и управления, проектированию информационных систем и улучшению бизнес-процессов организаций.

Библиогр.: 33 назв. Табл. 7. Рис. 24.

УДК 004.00(075.8)
ББК 32.97я73

ISBN 978-5-7996-3594-7

Оглавление

Предисловие 4
1. Понятия, модели и методологии жизненного цикла информационных систем 5
1.1. Основные понятия, связанные с жизненным циклом информационной системы 5
1.2. Структуры жизненного цикла 7
1.3. Модели и методологии жизненного цикла разработки 16
1.4. Выбор оптимальной модели и методологии ЖЦ 41
1.5. Информационная поддержка жизненного цикла: работа в *Trello* 45
2. Постановка проблемы и экономическое обоснование ИТ-проекта 52
2.1. Постановка проблемы и целей ИТ-проекта 53
2.2. Экономическое обоснование проекта разработки и внедрения ИС 60
Контрольные вопросы 81
Список библиографических ссылок 83
Приложение 1. Практические задания 87
Приложение 2. Критерии оценивания выполнения практических заданий 92

Предисловие

Пособие предназначено для студентов, изучающих дисциплины, связанные с проектированием и разработкой информационных систем. Его особенностью является сочетание теоретического материала с практическими заданиями, даны примеры их решения и критерии получения итоговой оценки по дисциплине.

В первой главе рассмотрен жизненный цикл информационных систем, его основные модели и этапы, с точки зрения различных российских и международных стандартов, приведено описание методологий разработки ПО как центрального этапа создания любой информационной системы.

Вторая глава содержит описание и примеры одного из вариантов осуществления первых этапов ИТ-проекта по разработке информационной системы, начиная от обнаружения проблемы, которую можно решить разработкой или внедрением ИС, заканчивая экономическим обоснованием данного решения.

В прил. 1 содержатся практические задания, позволяющие студентам, обучающимся по системе проектного обучения, выполнить междисциплинарный проект, который в свою очередь может стать основой для подготовки выпускной квалификационной работы.

В прил. 2 приведены критерии оценки выполнения практических заданий.

1. Понятия, модели и методологии жизненного цикла информационных систем

Информационная система — сложный объект, имеющий свой жизненный цикл, в течение которого протекают различные процессы, обеспечивающие функционирование этого объекта. Для того чтобы информационная система эффективно работала, удовлетворяла основные потребности пользователя, необходимо изучать ее на всех этапах. В этой главе вводятся основные понятия, относящиеся к информационной системе и ее жизненному циклу, описываются наиболее популярные модели жизненного цикла информационной системы, методологии, используемые при разработке.

1.1. Основные понятия, связанные с жизненным циклом информационной системы

ГОСТ Р 57193–2016 «Системная и программная инженерия. Процессы жизненного цикла систем», основанный на международном стандарте ISO/IEC/IEEE 15288:2015 *Systems and software engineering — System life cycle processes*, вводит следующее определение *Жизненного цикла*: «*Жизненный цикл (life cycle)* — развитие системы, продукции, ус-

луги, проекта или другой создаваемой человеком сущности от замысла до списания» [1]. В ГОСТах (Р 50–605–80–93) также уточняется, что «*жизненный цикл* — это не временный период существования, а процесс последовательного изменения состояния, обусловленный видом производимых воздействий» [2].

Жизненный цикл (ЖЦ) программного обеспечения (ПО) имеет определенные особенности по сравнению с понятием ЖЦ системы. IEEE *standard glossary of software engineering terminology* (IEEE Std 610 12–1990) подчеркивает, что следует различать жизненный цикл ПО и цикл разработки ПО. *Жизненный цикл ПО*, согласно стандарту IEEE, определяется как «период времени, который начинается, когда программный продукт только задумывается, и заканчивается, когда программное обеспечение больше не доступно для использования» [3]. ЖЦ ПО обычно включает этапы: концепция, требования, проектирование, реализация, тестирование, установка и проверка, эксплуатация и обслуживание и иногда — этап вывода из эксплуатации. Эти этапы могут перекрываться или выполняться итеративно [3].

В этом же стандарте *жизненный цикл разработки ПО* определяется как «период времени, который начинается с принятия решения о разработке программного продукта, и заканчивается, когда программное обеспечение поставляется» [3]. Этот цикл обычно включает этапы: требования, проектирование, реализация, тестирование, а иногда и этап установки и проверки. Перечисленные этапы могут перекрываться или выполняться итеративно, в зависимости от используемого подхода к разработке программного обеспечения. Кроме того, этот термин иногда используется для обозначения более длительного периода времени, либо «периода, который заканчивается, когда программное обеспечение больше не улучшается разработчиком, либо периода всего жизненного цикла программного обеспечения» [3].

Из этих определений следует, что ЖЦ программного обеспечения включает в себя ЖЦ разработки программного обеспечения.

Вопросы для самоконтроля к главе 1.1

1. Дайте определение понятию жизненный цикл информационной системы.
2. В чем отличие жизненного цикла программного обеспечения и жизненного цикла разработки программного обеспечения?

1.2. Структуры жизненного цикла

Рассмотрим основные разновидности структур жизненного цикла.

Классическая структура жизненного цикла

Одним из первых жизненный цикл информационной системы описал Уинстон У. Ройс в статье *Managing the Development of Large Software Systems* [4]. Этот цикл представлен на рис. 1, и именно он считается классическим.

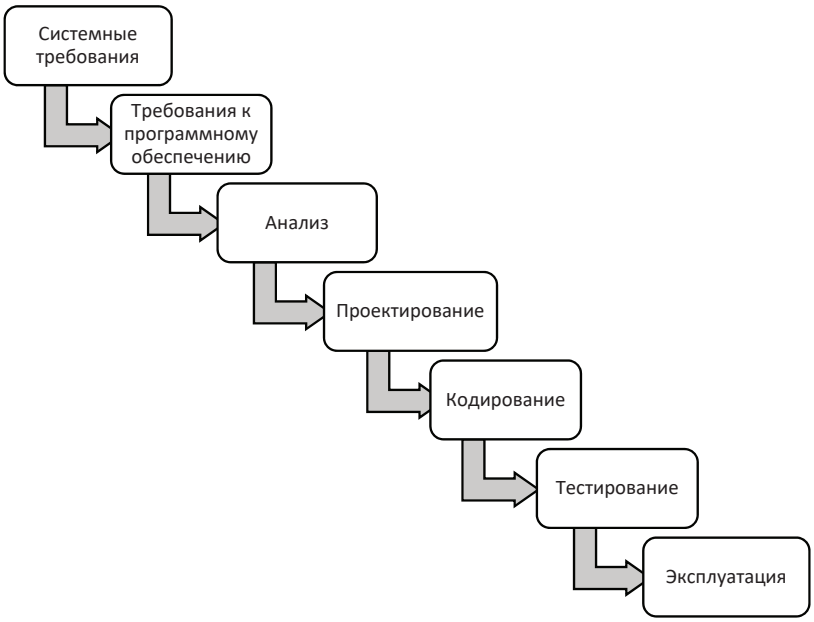


Рис. 1. Классический ЖЦ ИС по У. Ройсу [4]

Особенностью данной структуры является то, что переход от одной стадии к другой осуществляется строго последовательно.

Подход на основе ГОСТов (19 и 34 серии)

Существуют (и до сих пор считаются действующими) две серии ГОСТов — 19 и 34, которые определяют стадии и документирование ЖЦ ПО. Кроме того, введен в действие ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная

инженерия. Процессы жизненного цикла программных средств», основанный на международном стандарте ISO/IEC 12207:208 *System and software engineering — Software life cycle processes*.

19 серия — это «единая система программной документации», а 34 — «комплекс стандартов на автоматизированные системы». Отличительной чертой обеих серий стандартов является определение перечня документов, которые должны быть разработаны в процессе создания информационной системы. В ГОСТах 19 серии все основные стадии называются так, как называется основной документ отчетности. ГОСТ 19.102–77 разбивает ЖЦ на стадии: техническое задание, эскизный проект, технический проект, рабочий проект и внедрение. ГОСТы этой серии в основном охватывают ЖЦ разработки информационной системы.

Эти серии стандартов не противоречат друг другу, поэтому рассмотрим более подробно 34 серию стандартов как более новую и поддерживающую системный подход.

ГОСТ 34.601–90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания» определяет стадии создания информационной (автоматизированной) системы, представленные ниже [5].

1. Формирование требований к автоматизированной системе (АС):
   - обследование объекта и обоснование необходимости создания АС;
   - формирование требований пользователя к АС;
   - оформление отчета о выполнении работ и заявки на разработку АС.
2. Разработка концепции АС:
   - изучение объекта;
   - проведение необходимых научно-исследовательских работ;
   - разработка вариантов концепции АС и выбор варианта концепции АС, удовлетворяющего требованиям пользователей;
   - оформление отчета о проделанной работе.
3. Техническое задание:
   - разработка и утверждение технического задания на создание АС.
4. Эскизный проект:
   - разработка предварительных проектных решений по системе и ее частям;
   - разработка документации на АС и ее части.

5. Технический проект:
    - разработка проектных решений по системе и ее частям;
    - разработка документации на АС и ее части;
    - разработка и оформление документации на поставку комплектующих изделий;
    - разработка заданий на проектирование в смежных частях проекта.
6. Рабочая документация:
    - разработка рабочей документации на АС и ее части;
    - разработка и адаптация программ.
7. Ввод в действие:
    - подготовка объекта автоматизации;
    - подготовка персонала;
    - комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями);
    - строительно-монтажные работы;
    - пусконаладочные работы;
    - проведение предварительных испытаний;
    - проведение опытной эксплуатации;
    - проведение приемочных испытаний.
8. Тестирование АС.
9. Сопровождение АС:
    - выполнение работ в соответствии с гарантийными обязательствами;
    - послегарантийное обслуживание.

ГОСТ 34.601–90 также уточняет, что «технические проекты и рабочая документация — это последовательное построение все более точных проектных решений. Допускается исключать стадию „Эскизный проект“ и отдельные этапы работ на всех стадиях, объединять стадии „Технический проект“ и „Рабочая документация“ в „Технорабочий проект“, параллельно выполнять различные этапы и работы, включать дополнительные» [5].

Как можно заметить, в этом ГОСТе указывается на то, что каждый этап должен быть задокументирован, причем есть отдельные стандарты на оформление этой документации.

В ГОСТах 34 серии существуют 2 основных стандарта по документированию:

- ГОСТ 34.602–89 «Техническое задание на создание автоматизированной системы» (самый популярный стандарт на разработку технического задания (ТЗ), связан с другими стандартами этой серии) [6];
- ГОСТ 34.201–89 «Виды, комплектность и обозначения документов при создании автоматизированных систем» (базовый документ, в котором приводится полный перечень документации ГОСТ 34, рекомендации по кодированию документов, информация о том, к каким стадиям проекта относятся документы, а также как их можно объединить между собой) [7].

Часто упоминается еще один ГОСТ: РД 50-34.698-90 «Автоматизированные системы. Требования к содержанию документов» — довольно объемный стандарт с различной степенью детальности, описывающий содержание проектных документов, но на сегодняшний день он считается недействующим.

Достоинства и недостатки подхода с использованием ГОСТов

Основной недостаток использования подхода к интерпретации ЖЦ информационной системы на основе ГОСТов 34 и 19 серий — это то, что ГОСТы изданы в 90-е годы прошлого века и уже частично устарели. В качестве примера можно привести устаревшие требования в отношении архитектуры ИС [8]:

1) рассматриваются только двухуровневые приложения (клиентская программа и сервер СУБД);
2) речь идет только о реляционных базах данных и их логической модели;
3) рассматривается только однооконный интерфейс;
4) отчеты в системе предполагаются к производству только в бумажном виде;
5) разрабатываемая программа должна быть ориентирована на решение задачи обработки информации, которая имеет четкий вход и выход и узко специализирована (не рассматривается парадигма объектно-ориентированного программирования).

В стандартах используются устаревшие термины, например, «машинограмма», «видеокадр», сам термин «автоматизированная система управления (АСУ)», вынесенный в заголовок стандарта, сейчас практически не используется.

Достоинства подхода на основе ГОСТов соответствуют основным достоинствам любых стандартов — это предоставление общего «языка» для обсуждения требований к ПО и их реализации, возможность обеспечения совместимости с другими системами. Эти стандарты показали свою жизнеспособность, они используются долгое время, а значит доказали свою полезность и применимость (*usability*), «у них есть четкая цель — максимально полно описать на бумаге сложную абстрактную сущность, которую представляет собой любая АСУ» [9].

К достоинствам этой серии ГОСТов можно отнести и то, что они дают определенную свободу, предоставляя возможность объединения или корректировки содержания определенных стадий и документов.

Процессный подход. ГОСТ Р ИСО/МЭК

ГОСТ Р ИСО/МЭК 12207–2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств» использует процессный подход к описанию ЖЦ ИС.

В аннотации к стандарту указано, что «данный стандарт, используя устоявшуюся терминологию, устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Стандарт определяет процессы, виды деятельности и задачи, которые используются при приобретении программного продукта или услуги, а также при поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов» [10]. Основной целью создания стандарта заявлено «представление определенной совокупности процессов, облегчающих связи между приобретающими сторонами, поставщиками и другими правообладателями в течение жизненного цикла программных продуктов» [10]. В отличии от стандартов 19 и 34 серий данный стандарт «не устанавливает требований к документации в части ее наименований, форматов, определенного содержания и носителей для записи» [10].

В связи с важностью понятия *процесса*, которое использовано в стандарте, уточним этот термин, взяв его из ИСО 9000:2005: «*Процесс (process)* — есть совокупность взаимосвязанных или взаимодействующих видов деятельности, преобразующих входы в выходы» [11]. В стандарте определены 7 групп процессов.

1. Процессы соглашения (2 процесса).
2. Процессы организационного обеспечения проекта (5 процессов).

3. Процессы проекта (7 процессов).
4. Технические процессы (11 процессов).
5. Процессы реализации программных средств (7 процессов).
6. Процессы поддержки программных средств (8 процессов).
7. Процессы повторного применения программных средств (3 процесса).

Каждый из процессов ЖЦ описывается в терминах цели и желаемых выходов, для него определены списки действий и задачи, которые необходимо выполнять для достижения этих результатов.

Процессы в стандарте также разделены на группы по типам: основные, вспомогательные и организационные (рис. 2).

Рис. 2. Процессы жизненного цикла ИС [12]

К основным процессам отнесены:

- приобретение (действия и задачи заказчика, приобретающего ПО);
- поставка (действия и задачи поставщика, который снабжает заказчика программным продуктом или услугой);

- разработка (действия и задачи, выполняемые разработчиком: создание ПО, оформление проектной и эксплуатационной документации, подготовка тестовых и учебных материалов и т. д.);
- эксплуатация (действия и задачи оператора — организации, эксплуатирующей систему);
- сопровождение (действия и задачи, выполняемые сопровождающей организацией, т. е. службой сопровождения) — внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

К вспомогательным процессам отнесены:

- документирование (формализованное описание информации, созданной в течение ЖЦ ПО);
- управление конфигурацией (применение административных и технических процедур на всем протяжении ЖЦ ПО для определения состояния компонентов ПО, управления его модификациями);
- обеспечение качества (обеспечение гарантий того, что ИС и процессы ее ЖЦ соответствуют заданным требованиям и утвержденным планам);
- верификация (определение того, что программные продукты, являющиеся результатами некоторого действия, полностью удовлетворяют требованиям или условиям, обусловленным предшествующими действиями);
- аттестация (определение полноты соответствия заданных требований и созданной системы их конкретному функциональному назначению);
- совместная оценка (оценка состояния работ по проекту: контроль планирования и управления ресурсами, персоналом, аппаратурой, инструментальными средствами);
- аудит (определение соответствия требованиям, планам и условиям договора);
- разрешение проблем (анализ и решение проблем независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения или других процессов).

К организационным процессам отнесены:

- управление (действия и задачи, которые могут выполняться любой стороной, управляющей своими процессами);

- создание инфраструктуры (выбор и сопровождение технологии, стандартов и инструментальных средств, выбор и установка аппаратных и программных средств, используемых для разработки, эксплуатации или сопровождения ПО);
- усовершенствование (оценка, измерение, контроль и усовершенствование процессов ЖЦ);
- обучение (первоначальное обучение и последующее постоянное повышение квалификации персонала).

Каждый процесс разделен на действия. Например, процесс приобретения охватывает следующие действия:

1) инициирование приобретения;
2) подготовка заявочных предложений;
3) подготовка и корректировка договора;
4) надзор за деятельностью поставщика;
5) приемка и завершение работ.

Каждое действие включает ряд задач. Например, подготовка заявочных предложений должна предусматривать:

1) формирование требований к системе;
2) формирование списка программных продуктов;
3) установление условий и соглашений;
4) описание технических ограничений (среда функционирования системы и т. д.).

Особый интерес представляют технические процессы, именно в ходе выполнения этих процессов разрабатывается ИС. В стандарте приведен следующий перечень технических процессов:

1) определение требований;
2) анализ системных требований;
3) проектирование архитектуры системы;
4) процесс реализации программных средств;
5) процесс комплексирования системы;
6) процесс квалификационного тестирования системы;
7) процесс инсталляции программных средств;
8) процесс поддержки приемки программных средств;
9) процесс функционирования программных средств;
10) процесс сопровождения программных средств;
11) процесс изъятия из обращения программных средств.

Графически технические процессы, в ходе которых осуществляется разработка (первые 8 из списка), представлены на рис. 3.

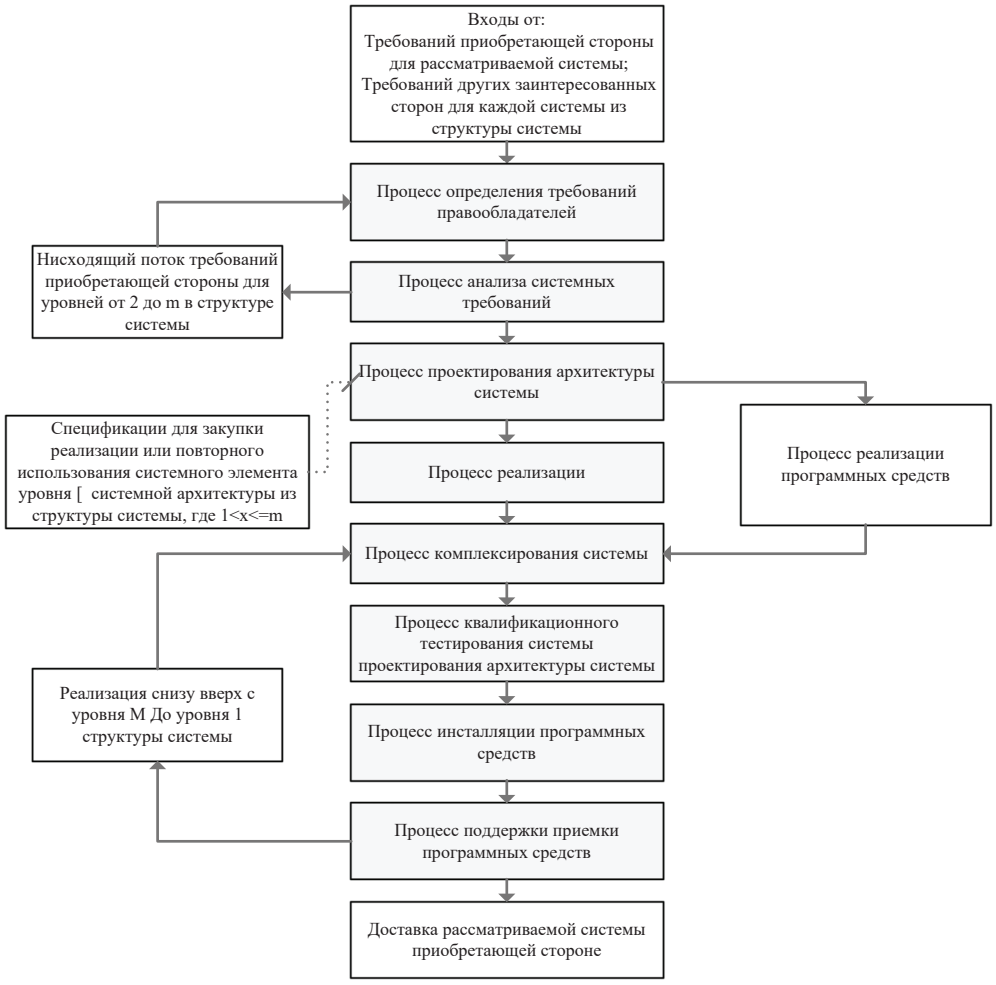


Рис. 3. Процесс реализации программных средств (ГОСТ Р ИСО/МЭК 12207–2010)

Из рис. 3 очевидно, что процессы необходимо согласовывать для получения эффективного результата.

Вопросы для самоконтроля к главе 1.2

1. Какой ЖЦ считается классическим для информационной системы? Кто автор этого подхода?
2. Из каких стадий состоит классический ЖЦ информационной системы?

3. Какие российские ГОСТы определяют стадии ЖЦ информационной системы, в чем их особенности?
4. В чем особенности процессного подхода при описании ЖЦ ПО?
5. На какие группы ГОСТ Р ИСО/МЭК 12207–2010 делит все процессы ЖЦ ИС?
6. Какие процессы обычно относят к техническим процессам ЖЦ ИС, и как они соотносятся со стадиями классического ЖЦ?

1.3. Модели и методологии жизненного цикла разработки

В основе любого жизненного цикла лежат модели и методологии, разработанные на их основе.

Понятие модели жизненного цикла

Модель ЖЦ (life cycle model) — структура процессов и действий, связанных с жизненным циклом, организуемых в стадии, которые также служат в качестве общей ссылки для установления связей и взаимопонимания сторон [10].

Под *стадией* в данном контексте понимается «часть процесса создания ПО, ограниченная определенными временными рамками и заканчивающаяся выпуском конкретного продукта (моделей, программных компонентов, документации), определяемого заданными для данной стадии требованиями» [10]. Стадии в модели могут перекрываться и (или) повторяться циклически в соответствии с областью применения, размером, сложностью, потребностью в изменениях и возможностях.

Кроме стадий модель ЖЦ включает: результаты выполнения работ на каждой стадии и ключевые события (точки завершения работ и принятия решений).

ГОСТ Р ИСО/МЭК 12207–2010 не требует использования какой-либо конкретной модели жизненного цикла. Однако он требует, чтобы «в каждом проекте определялась подходящая модель жизненного цикла, предпочтительно та, которая уже выбиралась организацией для применения в различных проектах. Применение модели жизненного цикла обеспечивает средства для установления зависимой от времени последовательности, необходимой для менеджмента проекта» [10].

На данный момент в литературе описаны различные типы или классы моделей жизненного цикла. Примеры этих типов моделей известны под такими наименованиями, как каскадная, пошаговая, эволюционная и спиральная разработка. В последнее время в связи с появлением экстремального программирования появился еще ряд моделей жизненного цикла.

1.3.1. Классические модели и методологии

Первыми появились модели и основанные на них методологии, описанные далее в этой главе и известные как классические.

Каскадная модель

Именно эта модель использована У. Ройсом при описании ЖЦ ИС и была представлена на рис. 1. Она имеет и другие названия — *модель водопада (waterfall), каскадная модель*. Особенностью модели является то, что переход к следующей стадии осуществляется только после того, как будет полностью завершена работа на предыдущей; возвратов на пройденные стадии не предусматривается. К достоинствам модели относятся:

- *устойчивость к изменению кадрового состава;* разработчики могут приходить и уходить на протяжении всего жизненного цикла проекта, но благодаря подробному документированию это практически не влияет на сроки исполнения проекта;
- *дисциплинирует;* заставляет разработчиков, вовлеченных в проект, быть дисциплинированными, оставаться в рамках намеченного плана. При необходимости в общей модели добавляется орган управления, ответственный за принятие решений, исполнители же обязаны работать в рамках системы;
- *гибкость на ранних этапах;* изменения в первых трех фазах могут быть сделаны немедленно и с минимальными усилиями, поскольку они не подкреплены кодом, что позволяет заказчику и исполнителю иметь значительный временной запас для кардинального изменения концепции работы ПО;
- *ориентация на сроки и финансы;* благодаря тому, что каждый этап полностью очерчивает контур будущего ПО, все разработчики понимают свою роль, границы работы и сроки исполнения, что позволяет оперировать реальными цифрами перед заказчиком и делает модель проекта привлекательной.

Недостатки модели:

- *неадаптивная структура ПО;* на первых этапах модель водопада может быть гибкой, но если на фазе тестирования выявляются проблемы в общей структуре — это влечет за собой плачевные последствия в виде сорванных сроков и даже отказов заказчика;
- *игнорирует конечного пользователя;* чем ниже продвигается процесс в водопаде, тем меньше в нем роль заказчика, не говоря уже о клиентах, которых он представляет. Внесение каких-либо изменений в функциональность ПО запускает всю цепочку этапов заново, поэтому продукты, полученные по каскадной модели, далеки от ориентации на массового пользователя;
- *позднее тестирование;* именно на этапе тестирования чаще всего выявляются ошибки, допущенные на каждом из этапов. Более гибкие методологии используют тестирование в качестве фундаментальной операции, происходящей непрерывно. «Водопад» же допускает низкую квалификацию сотрудников на каждом этапе и плохое качество исполнения, ведь при запоздалом тестировании проблемы невозможно решить фундаментально, только при помощи «заплаток».

Таким образом, каскадная методология — хорошее решение, с точки зрения сроков и отчетности, но очень слабое в плане качества, поэтому сегодня ее рекомендуется использовать только в трех случаях [13]:

1) при ориентации ПО на заказчика, требующего прозрачность работ и исполнение в назначенные сроки;
2) при наличии в штате руководителей соответствующей квалификации;
3) при исполнении проекта, не имеющего конкуренции на рынке.

V-модель

V-модель — это улучшенная версия классической каскадной модели, которая имеет специфику проектов для государственных органов: фиксированные требования, стоимость и время. В модели на каждом этапе происходит контроль текущего процесса, для того чтобы убедиться в возможности перехода на следующий уровень. Тестирование начинается еще со стадии написания требований, причем для каждого последующего этапа предусмотрен свой уровень тестового покрытия [14].

Для каждого уровня тестирования разрабатывается отдельный тест-план, т. е. во время тестирования текущего уровня мы также занима-

емся разработкой стратегии тестирования следующего. При создании тест-планов определяются ожидаемые результаты тестирования и указываются критерии входа и выхода для каждого этапа.

В *V*-модели каждому этапу проектирования и разработки системы соответствует отдельный уровень тестирования. Здесь процесс разработки представлен нисходящей последовательностью в левой части условной буквы *V*, а стадии тестирования — на ее правом ребре (рис. 4). Соответствие этапов разработки и тестирования показано горизонтальными линиями.

Рис. 4. *V*-модель жизненного цикла

Достоинства *V*-модели:

- строгая последовательность этапов;
- планирование тестирования и верификация системы производятся на ранних этапах;
- улучшенный, по сравнению с каскадной моделью, тайм-менеджмент;
- промежуточное тестирование.

Недостатки *V*-модели:

- недостаточная гибкость модели;
- создание программы происходит на этапе написания кода, т. е. уже в середине процесса разработки;

- недостаточный анализ рисков;
- нет работы с параллельными событиями и возможности динамического внесения изменений.

Когда рекомендуется использовать *V*-модель:

1) в проектах, в которых существуют временные и финансовые ограничения;
2) для задач, которые предполагают более широкое, по сравнению с каскадной моделью, тестовое покрытие.

Пошаговая (инкрементная) модель

В начале работы над проектом определяются все основные требования к системе, после чего выполняется ее разработка в виде последовательности версий. При этом каждая версия является законченным и работоспособным продуктом. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система, так образуется *пошаговая,* или *инкрементная модель*, показанная на рис. 5.

Применение данной модели ЖЦ характерно для разработки сложных и комплексных систем, для которых имеется четкое видение того, что собой должен представлять конечный результат — информационную систему. Разработка версиями ведется в силу разного рода причин:

- отсутствия у заказчика возможности сразу профинансировать весь проект;
- отсутствия у разработчика необходимых ресурсов для реализации сложного проекта в сжатые сроки;
- требований поэтапного внедрения и освоения продукта конечными пользователями, в случае если внедрение всей системы сразу может вызвать у ее пользователей неприятие и только затормозить процесс перехода на новые технологии.

Достоинства и недостатки этой модели аналогичны тем, что были отмечены для каскадной модели, но в отличие от каскадной модели заказчик может раньше увидеть результаты. Уже по результатам разработки и внедрения первой версии он может незначительно изменить требования к разработке, отказаться от нее или предложить разработку более совершенного продукта с заключением нового договора.

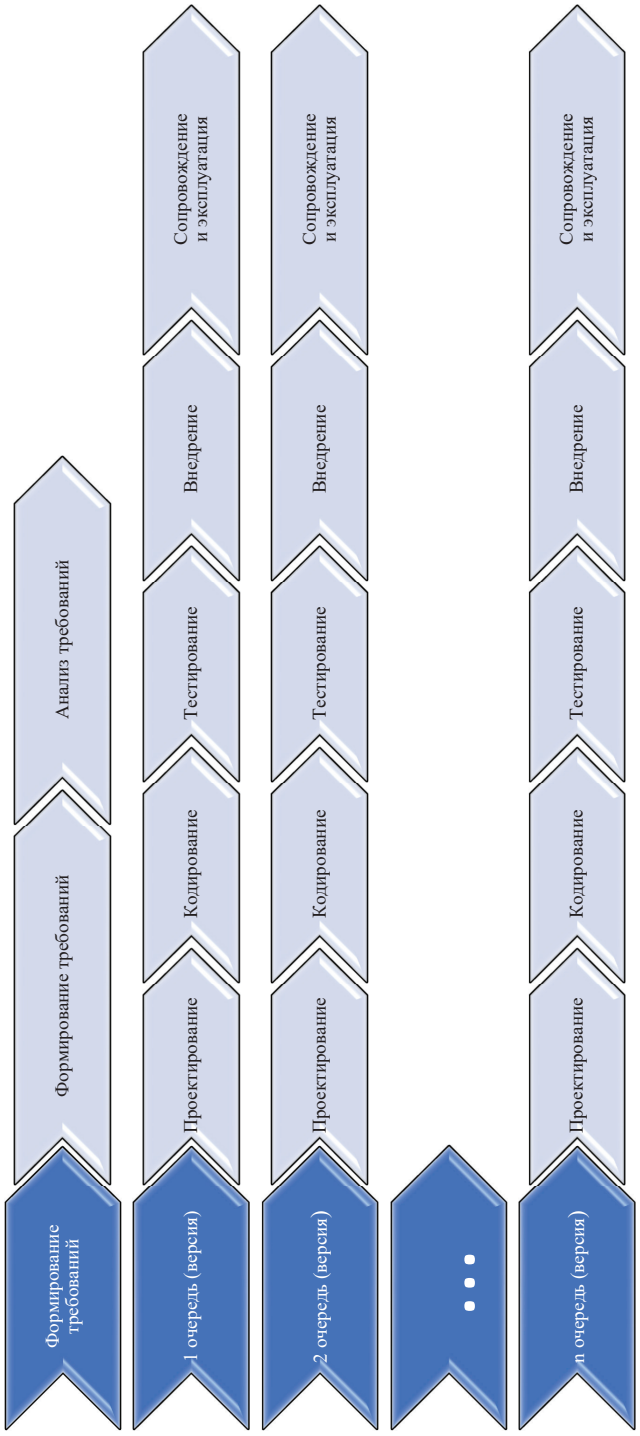


Рис. 5. Пошаговая (инкрементная) модель жизненного цикла [12]

Спиральная модель

Спиральная модель (эволюционная, или *итерационная модель)* автора Барри Боэм, 1986–1988 гг., подразумевает разработку в виде последовательности версий, но в начале проекта определены не все требования. Требования уточняются в результате разработки версий.

Данная модель жизненного цикла характерна при разработке новаторских (нетиповых) систем. В начале работы над проектом у заказчика и разработчика нет четкого видения итогового продукта (требования не могут быть четко определены) или нет уверенности в успешной реализации проекта (риски очень велики). В связи с этим принимается решение разработки системы по частям с возможностью изменения требований или отказа от ее дальнейшего развития. Графически модель представлена на рис. 6.

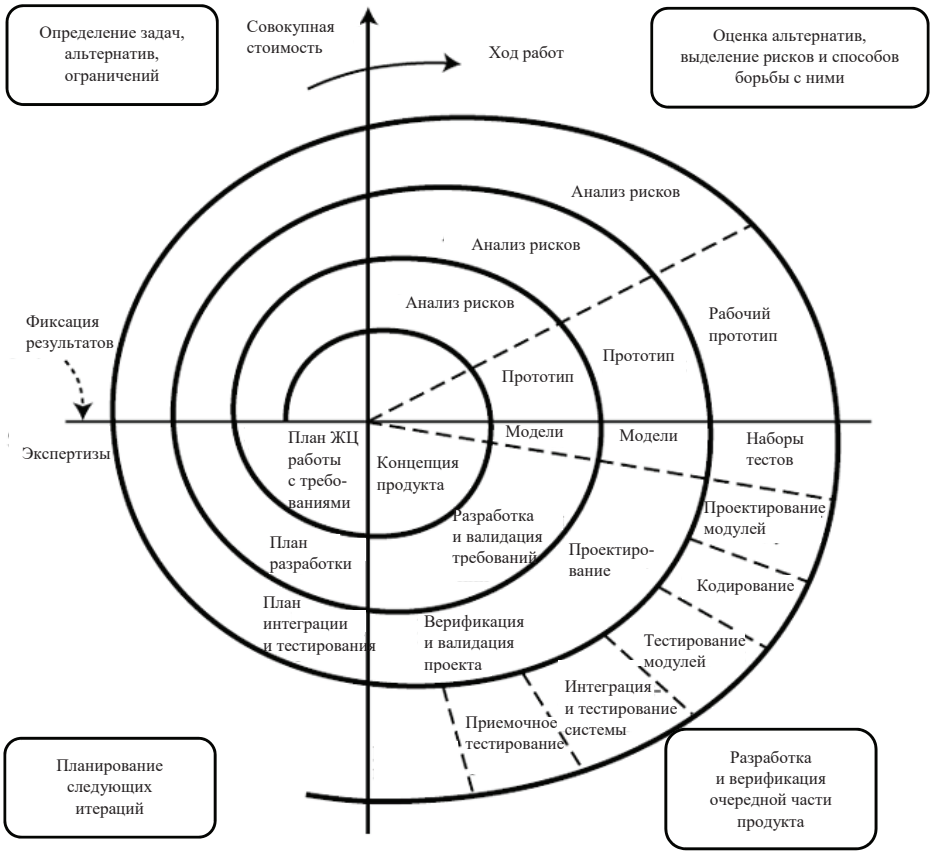


Рис. 6. Спиральная модель

Достоинства модели:

- позволяет быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований;
- допускает изменение требований при разработке системы;
- обеспечивает большую гибкость в управлении проектом;
- позволяет получить более надежную и устойчивую систему, т. к. по мере развития системы ошибки обнаруживаются слабые места и исправляются на каждой итерации;
- позволяет совершенствовать процесс разработки — анализ, проводимый в каждой итерации, позволяет провести оценку того, что должно быть изменено в организации разработки, и улучшить ее на следующей итерации;
- уменьшаются риски заказчика, т. к. заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта.

Недостатки модели:

- увеличивается неопределенность в перспективах развития проекта. Этот недостаток вытекает из предыдущего достоинства модели;
- затруднены операции временного и ресурсного планирования всего проекта в целом. Для решения этой проблемы необходимо ввести временные ограничения на каждую из стадий жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа выполнена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Методология RUP

В 2001 г. вышла методология разработки от компании IBM, называющаяся *Rational Unified Process* (RUP), которая описывает, как эффективно использовать коммерчески проверенные подходы к разработке программного обеспечения. RUP обеспечивает членов команды руководящими принципами, шаблонами и инструментальными наставлениями, необходимыми для всей команды, чтобы в полной мере воспользоваться следующими практиками [15]:

- разрабатывайте программное обеспечение итеративно;
- управляйте требованиями;

- используйте компонентную архитектуру;
- визуализируйте модель программы;
- проверяйте качество программы;
- контролируйте изменения программы.

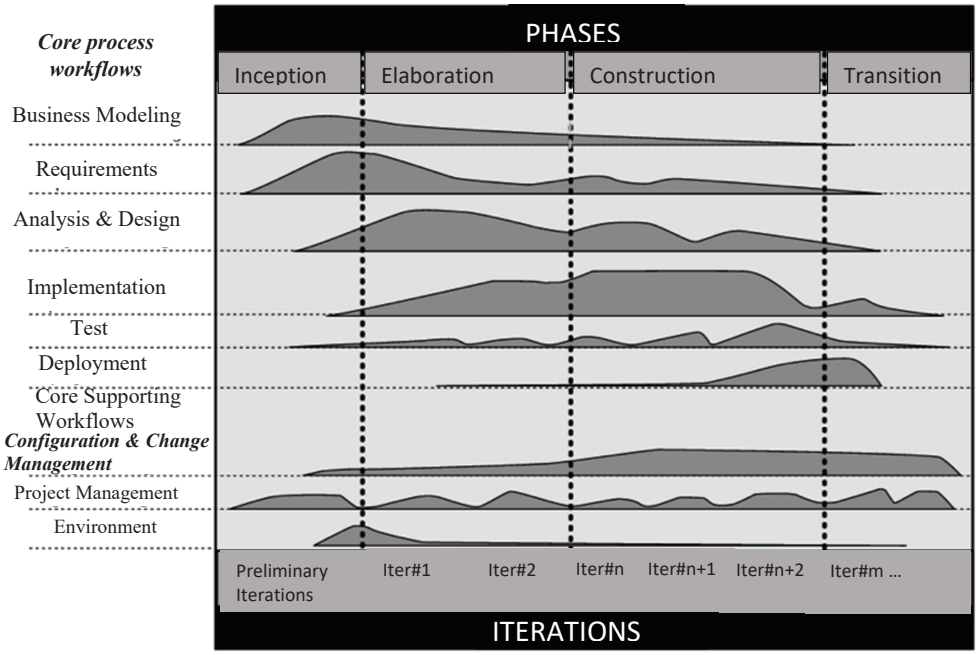


Рис. 7. Рабочие процессы RUP

Ядро RUP составляют следующие рабочие процессы, пересечение которых показано на рис. 7, среди которых 6 процессов инжиниринга и 3 вспомогательных процесса:

- бизнес моделирование (*Business Modeling*);
- требования (*Requirements*);
- анализ и дизайн (*Analysis & Design*);
- реализация (*Implementation*);
- тестирование (*Test*);
- развертывание (*Deployment*);
- управление проектами (*Project Management*);
- конфигурация и изменение (*Configuration & Change*);
- управление (*Management*);
- среда (*Environment*).

Методология RAD

Методология RAD (Rapid Application Development) ориентирована на быструю разработку приложения итеративно, с максимально простой архитектурой, минимальными издержками на процесс, максимально используя готовые компоненты и мощные инструменты. Имеет ограничение на длительность проекта — 60–90 дней.

RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки. Технология RAD предусматривает активное привлечение заказчика уже на ранних стадиях обследования организации, выработки требований к системе. Последнее из указанных свойств подразумевает полное выполнение требований заказчика как функциональных, так и нефункциональных с учетом их возможных изменений в период разработки системы, а также получение качественной документации, обеспечивающей удобство эксплуатации и сопровождения системы. Это означает, что дополнительные затраты на сопровождение сразу после поставки будут значительно меньше. Таким образом, полное время от начала разработки до получения приемлемого продукта при использовании этого метода значительно сокращается. На рис. 8 показано сравнение 2 моделей — на основе методологии RAD и каскадной модели.

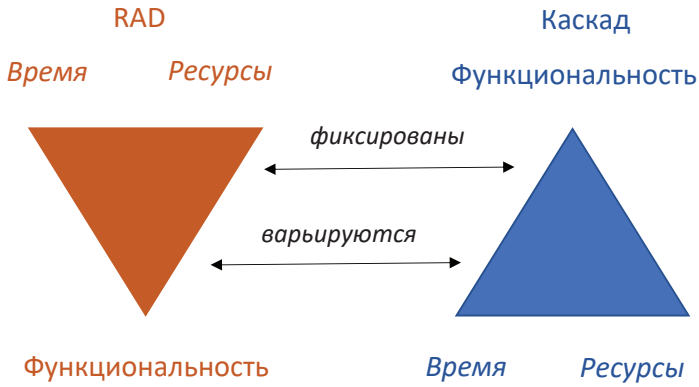


Рис. 8. Сравнение моделей ЖЦ ИС: каскадной и от методологии RAD

На основе этого сравнения, типа проекта и имеющихся ресурсов осуществляется выбор технологии.

1.3.2. Гибкие модели и методологии

В феврале 2001 г. семнадцать человек, ИТ-специалисты и разработчики, составили небольшой документ — *Agile*-манифест, который стал новацией в разработке программного обеспечения и положил начало ряду практических подходов к программированию [17].

С английского *agile* переводится как «подвижный, быстрый, проворный», но в русской ИТ-лексике за этой группой методологий закрепилось определение «гибкие». *Agile*-подход динамично организует программирование, постоянно адаптируя проект к новым обстоятельствам и требованиям.

В *Agile*-методологии в приоритете не исходные установки, а актуальные потребности пользователя. Постоянно вносить изменения в ТЗ, даже в самый разгар разработки, для *Agile* — норма. В гибкой методологии не предусмотрен предварительный генеральный план, напротив, программный продукт пишется практически экспромтом.

Разработка проходит через ряд циклов — *итераций*. Каждая итерация — это фактически отдельный проект, где разрабатываются фрагменты программы, улучшается функциональность, добавляются новые возможности.

Гибкая методология — это не алгоритм разработки, а набор идей и принципов, на которых основаны конкретные практические решения. Можно считать это особой философией, которая задает вектор, а не предписывает действия.

Четыре центральных идеи *Agile Manifesto* [16]:

- люди и взаимодействие важнее, чем процессы и инструменты;
- работающее ПО важнее, чем исчерпывающая документация;
- сотрудничество с заказчиком важнее, чем согласование условий контракта;
- готовность к изменениям важнее, чем следование первоначальному плану.

На рис. 9 в графическом виде приведено сравнение гибкой и каскадной методологий разработки.

Принципы *Agile* [16]:

1) задача высшего приоритета — регулярно и как можно раньше удовлетворять потребности заказчика, предоставляя ему программное обеспечение;

2) учитывать, что требования могут измениться на любом этапе разработки. Если изменения быстро вносятся в проект, заказчик может получить конкурентные преимущества;
3) выпускать версии готовой программы как можно чаще — с промежутком от двух недель до двух месяцев;
4) ежедневно вместе работать над проектом — разработчикам и заказчикам;
5) поручить работу мотивированным профессионалам. Обеспечить поддержку и условия, довериться им, и работа будет сделана;
6) общаться напрямую — это самый эффективный способ взаимодействия внутри команды и вне ее;
7) считать главным показателем прогресса работающий продукт;
8) поддерживать постоянный ритм работы — касается и разработчиков, и заказчиков;
9) уделять пристальное внимание техническому совершенству и качеству проектирования — это повышает гибкость проекта;
10) минимизировать лишнюю работу;
11) стремиться к самоорганизующейся команде — в ней рождаются наиболее эффективные и качественные решения;
12) всем участникам команды постоянно искать способы повышать эффективность работы.

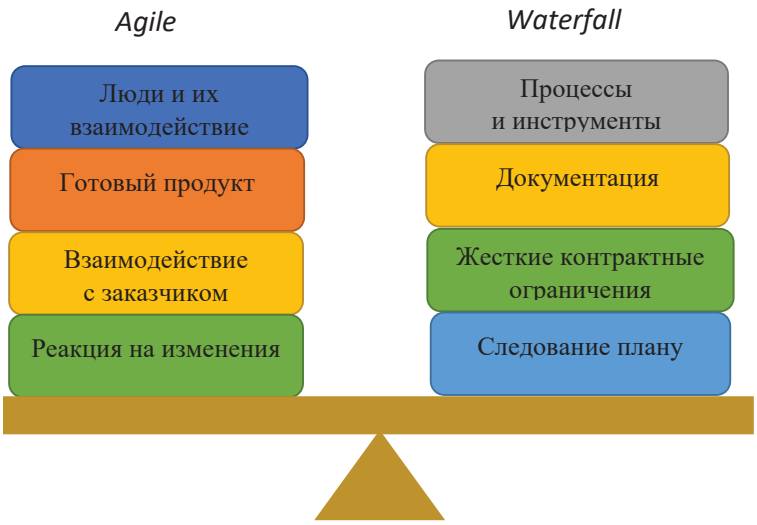


Рис. 9. Сравнение гибкой и классической методологий разработки

К сожалению, руководствуясь только этими идеями и принципами, выстроить рабочие процессы нельзя. По этой причине принято считать, что *Agile* — это класс, в рамках которого существует ряд прикладных методологий. На рис. 10 показана условная схема гибких подходов [18].

Рис. 10. Условная схема существующих гибких подходов [18]

Как видно из рис. 10, к гибким методам управления относятся фреймворк *Scrum* (Скрам) и метод *Kanban* (Канбан). Согласно исследованию *Agile* в России, Канбан сейчас занимает прочное второе место по популярности после Скрама [19].

Методология *Scrum*

Scrum появился до того, как сформировался манифест *Agile*, но его принципы хорошо укладываются в концепцию гибкой методологии, поэтому принято причислять его к этому семейству.

Впервые термин был использован в 1986 г. Японские исследователи Икуджиро Нонака и Хиротака Такеучи в статье *The new product development game* сформулировали принципы, позволяющие быстрее создавать новый продукт. Среди условий такой разработки назвали самоорганизующуюся команду специалистов, их полную свободу в творчестве и работе — без ограничений со стороны топ-менеджмента. Этот подход авторы описали так: «Это как оставить всех сотрудников

на втором этаже и убрать лестницу, а потом сказать: прыгайте или делайте что угодно — решение за вами. Экстремальные условия рождают творческий подход!» [20].

Руководство ставит перед командой задачу и, возможно, сообщает сроки, но не дает конкретных указаний, — рабочая группа самостоятельно ищет решение.

Главное для *Scrum*-подхода — особая динамика работы, когда команда постоянно обсуждает, как сделать продукт лучше. Такой ритм авторы сравнивают с регби: игроки передают мяч друг другу, но при этом команда движется по полю как единое целое, достигая общей цели. Из регби и пришел термин «скрам» — это схватка из-за мяча.

Благодаря Кену Шваберу и Джеффу Сазерленду *Scrum* пришел в сферу информационных технологий и приобрел популярность среди разработчиков.

Основные компоненты *Scrum* показаны на рис. 11.

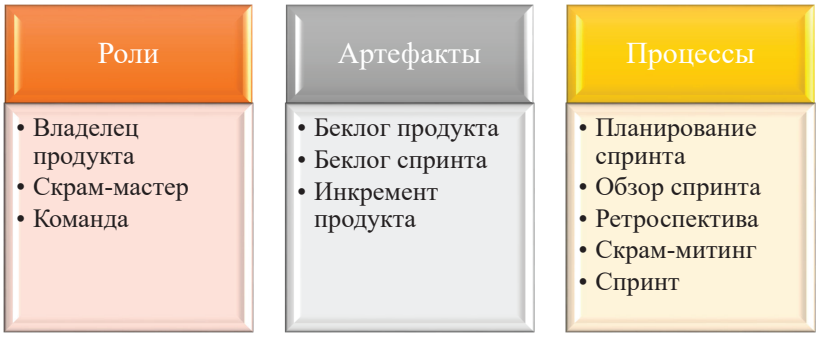


Рис. 11. Компоненты *Scrum*

В команде, работающей по принципам *Scrum*, нет внутренней иерархии: ни руководителей, ни подчиненных, ни указаний-приказов. Есть два особых члена группы: *product owner* — владелец продукта и *scrum master* — скрам-мастер.

Владелец продукта лучше всех знает, каким должен быть продукт, зачастую — это заказчик, его представитель или сотрудник, ответственный за взаимодействие с клиентом. Он должен ясно понимать, что именно требуется конечному пользователю программы. Все пожелания и предложения по функциональности и внешнему виду продукта (в *Scrum* они называются *stories* — истории) он заносит в специальный список — *Product Backlog* (беклог продукта).

На скрам-мастере лежит ответственность за сплоченную работу коллектива. Он не начальник команды, но делает все возможное, чтобы разработка шла в постоянном темпе, каждый участник был вовлечен и мотивирован, а важные детали не оставались без внимания.

Работа ведется спринтами — одинаковыми по продолжительности короткими итерациями. Все участники команды совместно планируют спринт, демонстрируют результаты заинтересованным лицам и ищут способы решения проблем, связанных с продуктом и процессом работы. В ходе спринта разработчики ежедневно обсуждают препятствия, краткосрочные планы и разделение работы между собой.

Беклог продукта (Product Backlog) — список требований с расставленными приоритетами и оценкой трудозатрат. Обычно он состоит из бизнес-требований, которые приносят конкретную бизнес-ценность и называются *элементы беклога*.

Беклог спринта (Sprint Backlog) — часть беклога продукта с самой высокой важностью и суммарной оценкой, не превышающей скорость команды, отобранная для спринта.

Инкремент продукта — новая функциональность продукта, созданная во время спринта.

Канбан

В общем смысле *Канбан* (англ. *Kanban* — вывеска, рекламный щит) — это система планирования для бережливого производства (также называемого производством «точно в срок», сокращенно — JIT (от англ. *Just-In-Time*)). Тайити Оно, промышленный инженер компании *Toyota*, разработал Канбан для повышения эффективности производства. Система получила свое название от карт, которые отслеживают производство на фабрике. Канбан также известен как система фирменных табличек *Toyota* в автомобильной промышленности.

Канбан-карты являются ключевым компонентом Канбана. Они сигнализируют о необходимости перемещать материалы внутри производственного объекта или перемещать материалы от внешнего поставщика на производственное предприятие. Карта Канбана, по сути, является сообщением, которое сигнализирует об истощении запасов продукта, запчастей или запасов. При получении сообщения Канбан запускает пополнение этого продукта, деталей или запасов. Таким об-

разом, потребление стимулирует спрос на увеличение производства, а канбан-карты сигнализируют о спросе на большее количество продукции, поэтому канбан-карты помогают создать систему, ориентированную на спрос.

В основе Канбана лежит простая мысль: объем незавершенной работы надо ограничивать. Любую новую задачу можно начинать не ранее, чем выполнена одна из начатых. Это не значит, что в работе должна быть только одна задача, — их может быть несколько, но принципиально, чтобы это количество было ограничено.

Чтобы контролировать выполнение задач было проще, процесс визуализируют на доске, разделенной на несколько колонок. В самом простом варианте это «в ожидании», «в работе» и «выполнено», но можно добавлять дополнительные, если необходимо детально отслеживать путь каждой задачи от постановки до реализации. На первых этапах лучше ограничиться необходимым минимумом.

Все задачи, которые поступают от заказчика, записываются на стикеры и размещаются в графе «в ожидании». Им можно назначить приоритет: более важные размещают выше и принимают в работу в первую очередь, а второстепенные — только когда будут выполнены приоритетные. Как только разработчик приступает к работе над очередной задачей, соответствующий листок переносится в графу «в работе». Когда она выполнена, стикер отправляется в последнюю графу.

Важное правило: количество задач-листков, находящихся в работе, ограничено. Точный максимум определяют исходя из возможностей коллектива. Взяться за следующую задачу разработчик сможет не раньше, чем закончит работу с предыдущей.

Важнейшее преимущество Канбана — наглядность. Видно не только то, какие задачи выполнены, находятся в разработке или ждут своего часа, но и насколько загружены разработчики. На доске можно выделить для каждого программиста или рабочей группы свою строку. Тогда канбан-карточки размещают в соответствующей колонке и строке, назначая задачу конкретным сотрудникам.

Методология Канбан имеет всего 3 правила:

1) визуализация процесса разработки с помощью канбан-доски;
2) ограничение на количество задач на каждом этапе;
3) постоянное измерение производительности команды и улучшения [21].

Сравнительные характеристики *Scrum* и *Kanban*

Далее представлено сравнение двух наиболее популярных гибких прикладных методологий — *Scrum* и *Kanban*:

Scrum	*Kanban*
Команда принимает участие в определенной итерации	Участие необязательно
Обязательна предварительная оценка	Оценка необязательна
Скорость используется для улучшения процесса	Для улучшения процесса используются временные ограничения
Задержка спринта только одной команды	Доска может быть поделена между несколькими командами
Используется минимум 3 роли (владелец, мастер, команда)	Ролей нет
Scrum-доска между спринтами меняется	*Kanban*-доска всегда неизменна
Для каждого спринта ставится приоритет	Приоритеты не являются обязательными

Поскольку *Scrum* и *Kanban* являются методиками *Agile*, то они могут применяться как отдельно друг от друга, так и дополнять одна другую. Ничто не мешает выбрать наиболее подходящее из каждой методики для своего проекта.

Методология экстремального программирования (*XP*)

Методология экстремального программирования (XP) ориентирована на разработку информационных систем группами малого и среднего размера в условиях неопределенных или быстро изменяющихся требований. Теория *XP* в своей основе проста: существует некий набор практик, применение которых способно дать лучший результат, нежели стандартные жесткие методики. Причем эти практики в *XP* применяются в превосходной экстремальной форме, откуда и название XP — *eXtreme Programming* [22].

Практики XP следующие:

1) планирование;
2) тестирование;

3) парное программирование;
4) рефакторинги;
5) простой дизайн;
6) коллективное владение кодом;
7) постоянные (непрерывные) интеграции кода;
8) заказчик в команде;
9) частые выпуски версий;
10) 40-часовая рабочая неделя;
11) стандарты кодирования;
12) метафора системы.

Рассмотрим эти практики подробнее.

Планирование — это та стадия, на которой обсуждаются требования с заказчиком. Необходимо получить наиболее полную информацию, поэтому обсуждения необходимо вести с профессионалом, знающим область, для которой создается продукт, а также всю операционную деятельность, для которой этот продукт предназначен. Обсуждение идет по принципу «от общего к частному»: сначала обсуждается задача в общих чертах, потом она разбивается на подзадачи. Когда все требования таким образом сформированы — они фиксируются где-нибудь в письменном виде. Затем заказчик определяет приоритеты.

После того как приоритеты расставлены, производится временная оценка. Нужно учитывать следующее: оценивать задачу может лишь тот, кто будет ею непосредственно заниматься, т. к. разработчики не идентичны и, кроме того, работают в разных ситуациях.

В *XP* принят такой подход: оценка работы при условии, что разработчик будет сконцентрирован исключительно на задаче, его ничто не будет отвлекать и он будет работать с максимальной продуктивностью — эта оценка называется *идеальное время*. Время, потраченное на разработку в действительности, называется *реальным*. Отношение этих времен, усредненное за определенный промежуток времени, называется *коэффициентом загрузки* (*load factor*). Таким образом разработчики оценивают идеальное время выполнения задачи, после чего, умножив его на коэффициент загрузки, получают реальную оценку выполнения задачи. Именно эта оценка должна фигурировать в описании задачи.

Принято считать, что коэффициент загрузки в начале работы команды равен 3. После того, как команда уже втянулась в проект, скорость разработки начинает повышаться. Для профессиональной команды,

хорошо сработавшейся друг с другом и с заказчиком, коэффициент загрузки по опыту находится где-то в районе 1.7–1.8.

При работе с заказчиком удобно иметь систему общего доступа, в которой будут отражаться все детали, касающиеся разработки. Это позволяет оперативно вносить изменения в работу. В качестве такой системы можно рассмотреть *TWiki*. В *TWiki* создается так называемая *история* (*user story*), в которой описывается все, что относится к конкретной задаче.

Стоит упомянуть еще один момент, связанный с планированием: если какая-то функциональность вызвала затруднения в реализации и все, что запланировано, не укладывается в рамки очередного релиза — объем реализуемой функциональности необходимо снизить (не отодвинуть выпуск продукта, а действительно уменьшить функциональность). Решить это должен исключительно заказчик. Очень часто часть функциональности таким образом безвозвратно отсеивается, что означает, что она изначально не была нужна.

В результате процесса планирования/перепланирования получается четкий план работы на ближайшую версию-две. Таким образом, планирование нужно производить по меньшей мере раз в версию, а лучше — раз в неделю, для того чтобы скорректировать оценки, основываясь на текущем положении вещей.

Тестирование. Под тестированием в данном случае понимается написание тестов перед кодированием. Тесты необходимы, т. к. в каждый момент времени, выполнив все тесты, можно утверждать, что система работает так, как от нее ожидают. Очень часто это помогает при каких-то внутренних изменениях, которые не должны затрагивать функциональность, например при рефакторингах.

Написание тестов до функциональности, во-первых, гарантирует, что любая часть функциональности системы покрывается тестами (если отложить тесты, появится большая вероятность того, что они так и не будут написаны), во-вторых, при написании тестов до функциональности система рассматривается как черный ящик с набором требований — на входе А, на выходе — В. Тест состоит в том, чтобы подать на вход А и сравнить результат с В. На практике тесты бывают существенно более сложными, с проверкой множества правил бизнес-логики.

Парное программирование означает работу двух разработчиков за одним компьютером, т. е. написание кода вдвоем. Это позволяет учесть

различные мнения и с большой вероятностью получить в каждой точке ветвления разные направления движения, т. е. получить две идеи вместо одной с возможностью выбрать более оптимальную из них. Если такое происходит часто, то оптимальность и ясность кода резко возрастают, уменьшается количество ошибок, что немаловажно для стоимости дальнейшей поддержки кода.

Кроме того, написание кода вдвоем означает, что код досконально знает не один человек, а два, таким образом знания о работе кода сложнее потерять. Третье преимущество: при работе в паре разработчики фактически подстегивают друг друга, в результате чего повышается скорость работы каждого из них. Однако это же является и минусом парной работы: работать вдвоем в таком режиме 8 ч в день — очень тяжело. Довольно часто парное программирование настолько пугает заказчика, что от *XP* отказываются только поэтому. В таком случае можно применять так называемое *параллельное программирование*. Отличается от парного оно исключительно тем, что у разработчиков не один компьютер, а два.

Рефакторинги. По определению *рефакторинг* (англ. *refactoring*), или перепроектирование кода, переработка кода, равносильное преобразование алгоритмов — процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения [23].

Необходимость в рефакторинге возникает потому, что программное обеспечение — вещь непостоянная, подверженная изменениям: требуется новая функциональность, расширяется старая. Каким бы тщательным ни было проектирование приложения — невозможно предугадать, что понадобится через полгода-год. Более того, попытки предугадать приведут лишь к пустой трате времени. Рефакторинг состоит в переработке системы для приведения в соответствие с текущими требованиями, при этом внешне система должна оставаться такой же.

В *XP* приняты так называемые постоянные рефакторинги, т. к. они жизненно необходимы в любой эволюционирующей системе.

Простой дизайн. Иногда говорят о простом дизайне и простом проектировании. Это означает, что при разработке всегда выбирается наиболее простое решение: «Лучше сделать простую вещь сегодня и завтра заплатить еще немного за внесение небольших изменений, чем делать сегодня сложную вещь, которая завтра может не понадобиться» [22].

Это правило неразрывно связано с правилом «непрерывных рефакторингов». Это как раз тот случай, когда результат больше, чем

простая сумма. Если писать просто, но не корректировать систему под текущие требования — хорошо не будет. Если корректировать, но писать при этом очень сложно — тоже ничего хорошего не получится, т. к. коррекция будет направлена как раз на то, чтобы исправить эти сложности. Если же писать просто и постоянно при этом корректировать код — результат будет гораздо лучше.

Коллективное владение кодом означает, что у кода нет автора и его может менять любой разработчик. Однако это не означает, что за код не отвечает никто. Любой, кто видит возможность улучшить какую-то часть кода, может сделать это в любой момент времени. Существует негласное правило: сломал — чини. Если изменения, внесенные кем-то, сделали код неработоспособным — именно автор изменений ответственен за приведение кода в рабочее состояние. Это подразумевает применение одинаковых стандартов и правил оформления кода с исчерпывающими комментариями, а также и ведение общедоступной истории развития системы.

Постоянная интеграция кода. При постоянной интеграции кода есть уверенность в том, что в любое время можно взять код и он будет работать. Или же есть информация о том, что код сломан, причем поступает она в самом худшем случае через час после того, как это случилось.

Заказчик в команде. Идея проста — чем длиннее цепочка, тем больше она напоминает испорченный телефон. Каким бы тщательным ни было планирование — некоторые вопросы все равно всплывут непосредственно в процессе реализации. И от того, насколько быстро и полно разработчик пообщается с заказчиком, зависит скорость реализации им задачи. Если отправить письмо, то придется ждать ответа. Часто неразрешенный вопрос блокирует дальнейшую работу, потому разработчик не сможет продолжить выполнение задач. Для максимальной эффективности заказчик (или его представитель, владеющий информацией и обладающий правом принимать решения) должен находиться в команде, чтобы вопрос можно было задать напрямую и достаточно быстро получить ответ.

Частые выпуски версий — очень важная практика по множеству причин. Чем меньше функциональности в очередном релизе, тем легче его протестировать. Если поменять половину системы, и в дальнейшем тестирование выявит плавающую ошибку — будет затрачена масса времени на ее поиск. Если в системе было одно единственное изменение — поиск практически бессмыслен. Кроме того, чем чаще

выпускается версия, тем быстрее заказчик получает требуемую функциональность. На каждую версию в результате планирования составляется список историй с приоритетами. После выпуска заказчик получает всю эту функциональность.

Сокращение временных затрат на получение функциональности — еще одно преимущество частых выпусков. Обычно за неделю до начала разработки очередной версии создается список историй, далее — одна итерация на разработку, неделю на тестирование, итого — две недели плюс длина итерации (чем короче, тем лучше). В стандартных методиках месяц уходит на техническое задание, два месяца на разработку и месяц на тестирование, итого — четыре месяца, что значительно дольше. Однако на самом деле через месяц в первом случае заказчик получит меньше, чем через четыре во втором. Заказчику в первом случае не приходится ждать четыре месяца, чтобы получить необходимую функциональность. Благодаря планированию самую необходимую ему часть он получает уже через месяц, а каждую следующую — через одну итерацию после предыдущей. Это и есть самый большой плюс частых выпусков версий.

В *XP* принято считать, что итерация должна занимать от недели до трех. Оптимально — две. Практика показывает, что за две недели можно успеть реализовать достаточно большие фрагменты функциональности.

Следует повторить, что, если разработчики не успевают закончить в итерации все, что запланировано, — необходимо сократить объем функциональности, но *не переносить* выпуск очередной версии. Единственный случай, когда релиз может быть перенесен, — когда этого требуют тестировщики. Выпустить «недотестированный» продукт или продукт, содержащий ошибки, намного хуже, чем отложить выпуск на неделю.

40-часовая рабочая неделя. Производительность разработчиков сильно падает, если они вынуждены работать подолгу. Причем падает настолько, что за десять часов они делают столько, сколько раньше делали за шесть. Существует убеждение, что программист не может работать в день больше шести часов, после этого резко снижается как скорость, так и качество работы.

Стандарты кодирования. Качественный код, с позиции менеджера по качеству кода, это код, максимально приспособленный к поддержке, т. к. редки ситуации, когда поддержка не требуется. Затраты

на поддержку, как показывает практика, зачастую серьезно превышают затраты на разработку. Код пишут разные разработчики, каждый со своим стилем, видением мира и квалификацией. И в этих условиях важнейшую роль играют правила, при соблюдении которых вероятность получить хорошо приспособленный к поддержке код сильно повышается.

Четыре основных правила создания качественного кода:

1) следовать принятым соглашениям (например, соглашения по форматированию);
2) обрабатывать исключительные ситуации;
3) документировать код (писать комментарии в коде);
4) код должен быть легко читаемым.

Метафора системы. Метафора системы — это аналогия, позволяющая точнее понять и прочувствовать систему. Путем проведения параллелей между чем-то незнакомым и чем-то интуитивно понятным можно дать хорошее представление о незнакомом предмете. Главное, правильно подобрать аналогию.

Для подбора аналогий нужен опыт и некоторый талант. Обычно этим занимается тот, кто прорабатывает архитектуру, т. к. только он может представить систему целиком и грамотно создать ее метафору. Это весьма непростое занятие, требующее навыков как в архитектуре, так и в обучении.

В заключение рассмотрим ключевые направления, на которые нацелена методология *XP*:

- работа с заказчиком: выяснение того, что в действительности ему нужно, т. к. своевременное выяснение подробностей и планирование сберегает много сил, средств, времени и нервов;
- гибкая разработка: простой дизайн, частые выпуски версий, регулярное планирование — все это служит тому, чтобы максимально быстро и безболезненно реагировать на меняющиеся требования заказчика и оперативно реализовывать функциональность, наиболее критичную прямо сейчас;
- непрерывная работоспособность кода: постоянные интеграции кода и большое количество тестов — это все дает уверенность, что код работоспособен и работает правильно;
- упрощение поддержки кода: рефакторинги и стандарты кодирования облегчают дальнейшие изменения в коде и облегчают понимание кода всеми разработчиками;

- повышение скорости и качества разработки: парное программирование, коллективное владение кодом, заказчик в команде, 40-часовая рабочая неделя и метафора системы — эти практики делают разработку более быстрой и качественной.

Преимущества и недостатки гибких методологий

Таким образом, можно отметить следующие основные преимущества гибких методологий [24]:

- программный продукт готов к использованию на самых ранних этапах его разработки, пусть и не с полной функциональностью;
- разработчики постоянно в контакте с заказчиком и пользователем и всегда знают, что именно требуется от программы, могут оперативно реагировать на новые потребности пользователя и пожелания к продукту;
- нет жестких рамок, поэтому программный продукт постоянно изменяется и улучшается, становится эффективнее и привлекательнее для пользователя;
- заказчик и пользователь в этой схеме выступают не столько как инвесторы, сколько как партнеры и идейные вдохновители.

К недостаткам гибких методологий можно отнести [24]:

- отсутствие уверенности в том, что программа когда-нибудь будет завершена, т. к. после каждой итерации и у разработчика, и у пользователя будут возникать новые идеи, как сделать продукт еще лучше, — разработка грозит растянуться на годы;
- большинство участников проекта со стороны пользователей могут на ранних этапах сформулировать множество требований к программе и будут ожидать, что все они будут реализованы в ближайших итерациях;
- строительство без чертежей — отсутствие генерального плана, концепции программы, единой структуры.
- ритм работы в *Agile* очень напряженный, нововведения изобретаются на лету, реализовывать их надо быстро, реагировать моментально и действовать оперативно, нет времени обдумывать все аспекты, неторопливо взвешивать за и против.

Методологии класса *Agile* хорошо себя покажут, если:

- в команде работают опытные и квалифицированные специалисты, умеющие действовать сообща и помогать друг другу;

- у заказчика нет ясного представления, как должна выглядеть программа, но он готов участвовать в совместной работе, чтобы развивать и улучшать продукт;
- сфера применения продукта постоянно меняется, часто возникают новые потребности и задачи;
- нет конкретных сроков, к которым продукт должен быть завершен, и жестких ограничений бюджета;
- работоспособную программу необходимо получить как можно скорее.

Такие условия могут сложиться, например, при работе над ИТ-стартапом.

Вопросы для самоконтроля к главе 1.3

1. Что такое модель и зачем она создается?
2. Что такое модель ЖЦ информационной системы?
3. Какие типы моделей ЖЦ ИС используются в настоящее время?
4. Назовите недостатки и преимущества каждой из моделей ЖЦ ИС.
5. Какая модель рекомендуется к использованию при разработке новаторской ИС?
6. Как давно в сфере ИТ используются гибкие методологии разработки и какие основные идеи, заложенные в эти методологии?
7. Опишите алгоритм реализации гибкой методологии.
8. Какой документ регламентирует применение гибких методологий?
9. В чем основные отличия гибкой и классической методологий разработки?
10. Назовите основные компоненты *Scrum*.
11. Какие артефакты использует *Scrum*?
12. Какие роли есть в команде *Scrum*, а какие в команде, работающей по методологии Канбан?
13. Что такое канбан-доска? Для чего она нужна?
14. Какие техники использует экстремальное программирование?
15. В чем недостатки гибких методологий? Когда они дают максимальный эффект?

1.4. Выбор оптимальной модели и методологии ЖЦ

Содержимое главы основано на материалах статьи «Блок-схема выбора оптимальной методологии разработки ПО» [25].

Существует множество моделей ЖЦ и методологий их реализации, поэтому часто сложно определить, что именно лучше подойдет для проекта. На рис. 12 представлена блок-схема, которую можно использовать для выбора оптимальной методологии.

ИТ-стартапы характерны тем, что там все строится на энтузиазме. Далеко не всегда участники работают по 8 ч в день, и если им навязать какую-то методологию и поставить в определенные рамки, то либо весь энтузиазм угаснет, либо никто не будет соблюдать правила. В дальнейшем, когда все стабилизируется, можно перейти к использованию подходящей методологии.

Каждый проект имеет риски. Однако в данном случае имеются в виду риски настолько серьезные, что заранее неизвестно, можно ли будет в принципе реализовать систему. Если такие риски присутствуют, то, скорее всего, разработка начнется с прототипов, концептов, моделей и т. п., чтобы выяснить принципиальную возможность задуманного. В таком случае наиболее оптимальной моделью будет спиральная модель. Типичный пример применения этой модели — исследовательские проекты (но не только). Если таких серьезных рисков нет, то встает вопрос, будут ли требования меняться. В случае четких неизменных требований и небольшой длительности проекта выбор каскадной модели, в сравнении с итеративной моделью, даст меньше накладных расходов на процесс. Программистов ничто не будет отвлекать от написания кода: ни необходимость срочно исправлять баги из прошлой итерации, ни бесконечные митинги и релизы.

В случае длительных проектов каскадная модель будет плохо работать. Несмотря на то, что риски изменения требований отсутствуют, присутствуют технические риски. Например, в случае разработки некорректной архитектуры, выбора не тех технологий и инструментов и т. п., с большой долей вероятности это выяснится в конце проекта и времени на исправление может не остаться. Чем дольше длится проект, тем выше цена исправления ошибки. Если на начальном этапе рефакторинги и кардинальные изменения кода даются легко, то под конец проекта что-то исправить уже довольно сложно.

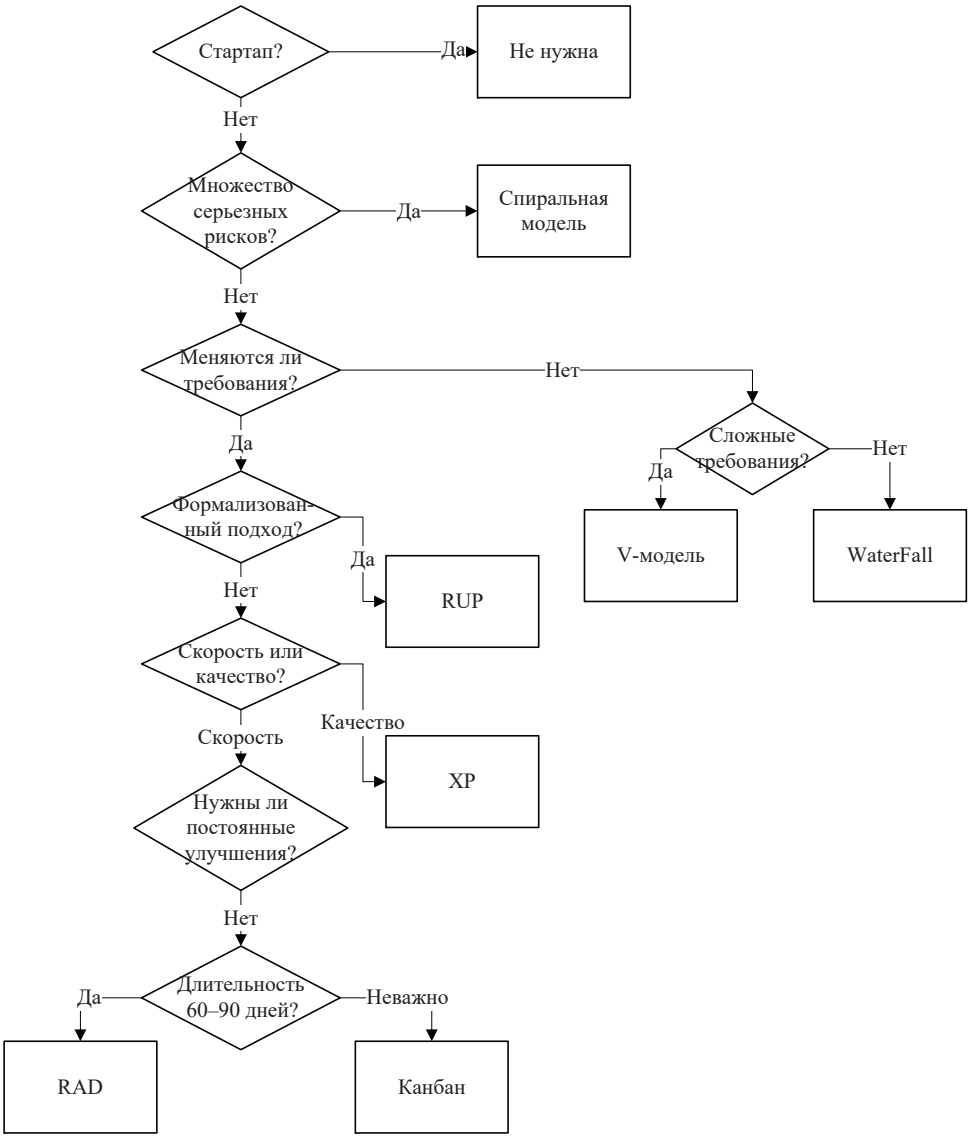


Рис. 12. Блок-схема выбора оптимальной технологии разработки ПО (по материалам [25])

Следующий вопрос, который следует задать для выбора жизненного цикла, касается сложности требований. Вопрос неоднозначный и зависит от проекта. На старте проекта бывает полезным на этапе анализа и проектирования продумать, как вы будете тестировать. Это может помочь раньше выявить потенциальные проблемы и лучше реализо-

вать архитектуру, проработать требования и т. п. Когда требования сложные, рекомендуется тщательно продумать все сценарии тестирования и еще на этапе анализа и проектирования разработать подходы к тестированию, тест-планы и тест-кейсы. В таком случае приходит на помощь *V-model* жизненного цикла.

Применение *V-model* там, где требования достаточно простые, приведет к тому, что система окажется дороже. Более того, *V-model* критикуют за то, что она порождает множество документации и бюрократии и служит не для того, чтобы создать качественный продукт, удовлетворяющий заказчика, а для того, чтобы в конце проекта доказать, что система работает, как и требовалось изначально.

Формализованный подход заключается в том, что все процессы жизненного цикла приложения детально регламентированы. Это необходимо в сложных больших проектах с большой командой. Примером могут служить системы, от которых зависят жизни людей: медицина, транспорт, энергетика. Обычно подобные системы разрабатываются в соответствии с отраслевыми стандартами, многократно тестируются, подлежат лицензированию. *Agile*-методологии в подобных системах не работают: в них живое общение предпочтительнее документации, но, например, если приложение будет тестироваться несколько раз разными командами, то лучше, если этот процесс будет тщательно задокументирован.

Таким образом, если требуется формализованный подход, то хорошим выбором станут такие методологии, как RUP. Такая методология больше, чем просто методология, и правильнее ее называть *фреймворком* процессов. Она изначально создана для итеративной модели, однако, ее можно путем модификаций использовать и в каскадной модели. Если же формализованный подход не нужен, то оптимально использовать *Agile*-методологии.

Еще один важный вопрос: скорость или качество (продуктивность или инженерия). Под продуктивностью понимается наиболее быстрый процесс добавления функционала в приложение. Под инженерией подразумевается высокий уровень организации разработки, новаторские подходы и сложные приемы, которые могут быть применены только опытной командой. Речь идет о таких практиках экстремального программирования, как разработка через тестирование, непрерывная интеграция, парное программирование и т. п. Особенность экстремального программирования заключается в том, что обязательно нужно

использовать все 12 практик, описанных в предыдущей главе, только тогда эффект от них становится максимальным. Если вы не будете использовать какую-то практику, то придется отказаться и от использования всех остальных.

Однако можно использовать практики экстремального программирования в других методологиях, более того, это может быть очень полезным. Например, можно использовать парное программирование в Скраме, чтобы помочь новичкам быстрее влиться в проект. Таким образом, экстремальное программирование может обеспечить высокое качество за счет более высокого уровня инженерии, но при выборе этой методологии проект может получиться дороже. Также для успешного применения обязательно требуется команда, уже имеющая опыт использования данной методологии.

Если же проекту нужна максимальная продуктивность, то есть другие варианты. Скрам ориентирован на постоянное усовершенствование процесса. Если в проекте опытная команда, хорошо отлаженный процесс и не нужны совершенствования, то следование методологии Скрам будет отнимать слишком много времени, которое можно потратить с большей пользой. Например, по Скраму, если спринт длится один месяц, то обзор спринта должен занимать 4 ч, ретроспектива спринта — 3 ч. Плюс к этому есть планирование спринта, длящееся 8 ч, и ежедневные Скрам-митинги по 15 мин.

Если в проекте неслаженная команда, используются или неопробованные технологии, или сфера применения приложения, или все вместе взятое, то выбор методологии Скрам может быть прекрасным решением. Когда начинается новый проект, можно начать использовать Скрам, а когда процесс станет более отлаженным, архитектура стабилизируется, потребность в постоянных улучшениях отпадет, то можно перейти к другой методологии, например Канбан.

Если совершенствования не нужны и все, что нужно, это сконцентрироваться на выполнении задач, то хорошо подойдет RAD или Канбан. RAD имеет много общего с *Agile*-методологиями, но в нем есть существенное ограничение на длительность проекта. Желательно не более 60–90 дней. Канбан похож на некий беспрерывный конвейер, который может длиться бесконечно. Он хорошо работает на проектах поддержки, но плохо там, где требуется разработать сложную архитектуру, т. к. ориентирован на быстрое обновление (добавление функциональности) приложения.

1.5. Информационная поддержка жизненного цикла: работа в *Trello*

Создание ИС осуществляется, как правило, в виде проектной разработки, поэтому информационная поддержка осуществляется инструментами управления проектами. Для управления проектной разработкой используют информационные инструменты различной сложности: от простых списков дел, или *to-do* листов, доступных любым пользователям, до сложных систем управления проектами с большим количеством функций, предназначенных для профессионалов в этой области.

Наиболее часто используются такие инструменты, как канбан-доски. В качестве примера рассмотрим *Trello* — информационную среду, которая принадлежит компании *Atlassian*.

Trello — это популярный инструмент управления, который позволяет: организовывать задачи, вести списки дел, просматривать инициативы, поддерживать обсуждения и визуализировать идеи, располагая весь этот функционал в одном месте. Сервис довольно прост и интуитивно понятен, и многим компаниям для работы достаточна его базовая бесплатная версия. Она подходит всем, кому нужна базовая функциональность досок Канбан. Имеет открытый API (т. е. можно написать расширение для этой системы) [26].

Интерфейс системы *Trello* представлен на рис. 13.

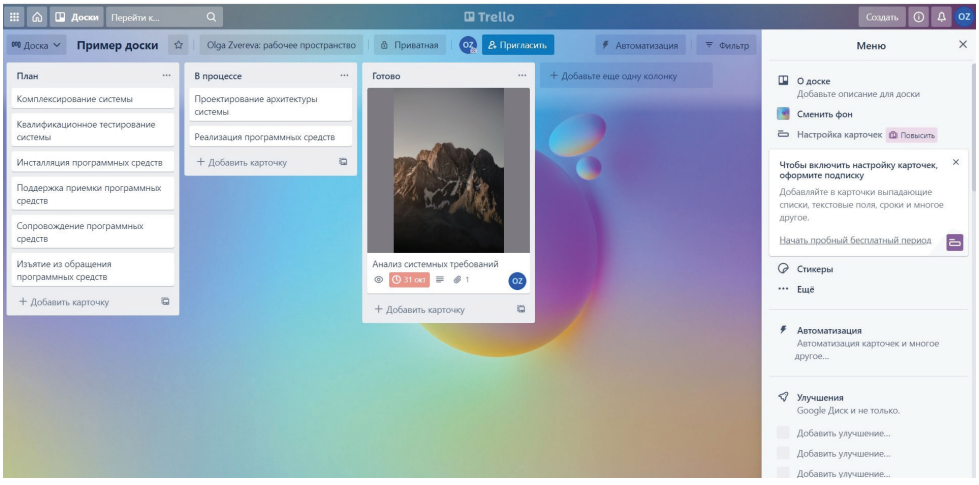


Рис. 13. Пример доски в *Trello*

Структура Trello. Три элемента, на которых держится структура организации проектов в приложении [27]:

- доска (*board*);
- список (*list*);
- карточка (*card*).

Доска — это один рабочий экран, который логически разделен на списки. Списки, в свою очередь, представляют собой вертикальные ряды для хранения карточек.

Карточки — это специальные формы для описания задач. Их можно двигать как внутри одного *списка*, так и свободно перемещать между списками или досками. Списки тоже можно перемещать. Для любой задачи можно назначить людей, ответственных за ее выполнение. *Trello* предлагает множество полезных возможностей для оформления, настройки и управления своими функциональными элементами. Каждая карточка может быть как простым описанием задачи, так и сложным документом со списками, чек-листами, вложениями, сроками, метками, ответственными лицами и т. д.

Что можно сделать с карточкой в *Trello*:

- переименовать, заполнить описанием и редактировать текст;
- присвоить метки, участников, срок выполнения, добавить файл или чек-лист;
- добавить комментарии, смайлы, вложения, другие задачи, оповестить выбранных участников;
- изменить положение блока в списке, перемещать его по спискам и другим доскам;
- скопировать, следить за изменениями, заархивировать;
- распечатать, экспортировать в формате *json*, поделиться ссылкой на карточку или ее почтовым адресом (письма будут появляться в виде комментариев);
- удалить навсегда.

Кроме этого, в самом низу у каждой задачи есть подробный лог: кто, когда и какие действия совершал. Вид карточки в *Trello* показан на рис. 14.

Списки, так же как карточки, можно копировать, перемещать и архивировать. Меню с досками в *Trello* можно сделать фиксированным, а сами доски добавлять в «избранные» и сортировать. Есть три типа досок с разным уровнем доступа:

- приватная (доступна только по личному приглашению владельца доски);

- командная (доступна всем участникам команды);
- публичная (может быть доступна всем).

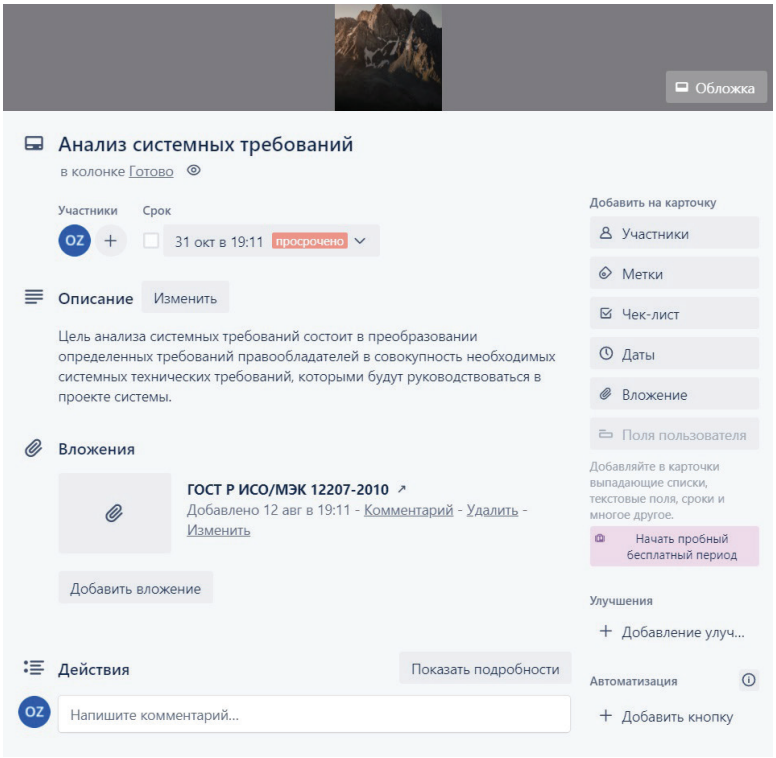


Рис. 14. Пример карточки в *Trello*

Закрытые доски и ненужные списки с карточками хранятся в специальном архиве (рис. 15). Оттуда их можно вернуть обратно или окончательно удалить. Можно создавать неограниченное количество задач, досок и списков, а также добавлять любое число участников.

Другие возможности Trello. Можно выделить несколько важных особенностей системы, которые позволяют работать более комфортно и гибко:

- встроенная «умная» система поиска с операторами и памятью запросов;
- наличие шаблонов (на рис. 16 показана доска, созданная на основе шаблона Канбан);
- кроме досок в *Trello* можно создавать персональные или бизнес-команды (по бизнес-подписке);

- *Trello* хранит подробные логи всех изменений и действий участников команды, есть возможность получать оповещения об изменениях в логе на почту или прямо на рабочий стол;
- есть удобный фильтр карточек по различным параметрам, он включает в себя настраиваемую систему цветных меток с режимом для дальтоников;
- открыть доступ к доске, карточке или пригласить человека в команду можно просто, скинув ему ссылку;
- для создания карточек можно делать закладки, перетаскивая гиперссылки с сайтов или прямо с компьютера.

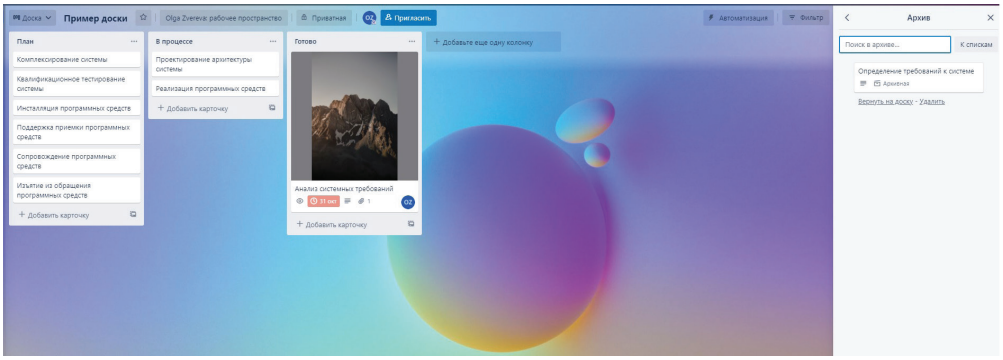


Рис. 15. Архив в *Trello*

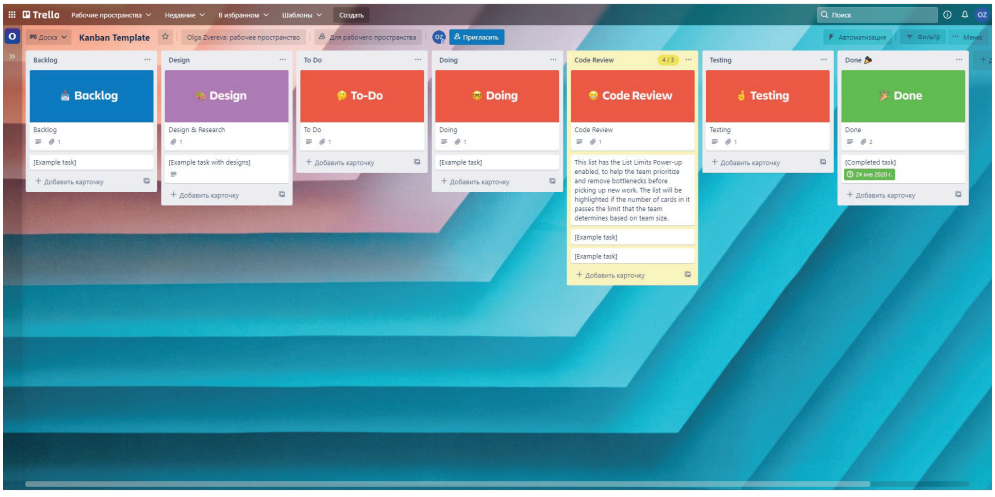


Рис. 16. Доска на основе шаблона Канбан

Советы по работе с Trello. Используйте горячие клавиши. Знание быстрых команд может существенно ускорить навигацию по *Trello*:

Команды	Горячие клавиши
Навигация по карточкам	←, ↓/*J*, ↑/*K*, →
Открыть меню досок в заголовке. Начать поиск	*B* */*
Отправить выбранную карточку в архив	*C*
Назначить выбранной карточке срок	*D*
Открыть режим быстрого редактирования карточки (курсор должен находиться на интересующей карточке)	*E*
Закрыть диалоговое окно и отменить все внесенные изменения	*Esc*
Сохранить набранный текст. Комбинация сработает, когда вы редактируете название, добавляете комментарии, меняете описание и проч.	*Control* + *Enter* (*Command* + *Enter*)
Открыть текущую карточку	*Enter*
Открыть меню фильтра карточек для удобного поиска	*F*
Открыть меню меток	*L*
Развернуть (или скрыть) названия меток на доске	;
Добавить новых участников или удалить существующих	*M*
Создание новой карточки	*N*
Перемещение карточек в соседние списки	<, >,. и,
Включение и отключение фильтра «карточки, назначенные мне»	*Q*
Подписаться на выбранную карточку (или быстро отменить подписку)	*S*
Добавить или удалить себя из списка участников карточки	*Space*
Открыть поле для переименования карточки	*T*

Добавить или удалить свой голос к карточке. Клавиша работает только при включенном улучшенном голосовании	*V*
Включить или отключить меню доски	*W*
Сброс всех активных фильтров	*X*
Открыть список горячих клавиш	?
Автоматическое дополнение списка участников карточки	@
Автоматическое дополнение списка меток для карточки	#
Автоматическое определение позиции для новой карточки	^

Добавляйтесь в участники к каждой важной задаче, чтобы получать уведомления об изменениях и держать процесс под контролем. Карточки, в которых вы значитесь как участник, появляются в отдельном списке в профиле.

Изучите такие стандартные улучшения, как «календарь», «поля пользователя», «голосование» и т. п. Переключение между ними даже в бесплатной версии может сделать работу с *Trello* еще комфортнее.

Разработайте собственную схему меток. Дайте им названия и продумайте структуру, которая поможет при необходимости фильтровать ваши задачи по срочности, важности, тематике, участникам или любому другому принципу.

Пользуйтесь облачными хранилищами данных: *Google* Диск, *Dropbox*, *OneDrive*, *Box*. Так вы сможете держать под рукой все нужные для проекта файлы, просто вставляя их в виде ссылок.

Пункты списка и элементы чек-листа в *Trello* можно превращать в карточки. Для этого каждый пункт списка должен начинаться с новой строки, после чего список нужно скопировать в окно для новых карточек и нажать *Enter*. Будет предложено разбить список на отдельные задачи или создать карточку с этим списком.

Скачайте и установите приложение для вашего мобильного устройства или расширение для браузера — это добавит новые функции *Trello*.

Достоинства и недостатки Trello. Одно из серьезных конкурентных преимуществ *Trello* перед другими системами управления проек-

тами — большой, постоянно обновляемый список улучшений (*Power-Ups*), разбитый на категории. Это расширения и интеграции с другими популярными сервисами, приложениями, облачными хранилищами данных и т. п.

Минус бесплатной версии *Trello* — это то, что можно включать ограниченное число улучшений для каждой доски, но между ними можно переключаться по необходимости. Найти приложения для интеграции можно перед «стикерами» в выпадающем меню справа.

К недостаткам *Trello* также можно отнести:

- функционала *Trello* недостаточно для действительно крупных компаний;
- на маленьких экранах *Trello* становится не таким удобным;
- автоматизацию, повторение и быстрое добавление задач сложно реализовать.

Поскольку в этой среде есть открытое API, то можно присоединиться к команде разработчиков и создать свое собственное расширение для *Trello*.

2. Постановка проблемы и экономическое обоснование ИТ-проекта

Вторая глава содержит последовательные описания одного из вариантов осуществления первых этапов ИТ-проекта по разработке информационной системы, начиная от обнаружения проблемы, которую можно решить разработкой или внедрением ИС, заканчивая экономическим обоснованием данного решения. Однако ИТ-проект может требовать иной последовательности, пропуска некоторых из описанных этапов или, наоборот, привлечения дополнительных этапов, которые при этом могут пересекаться или выполняться параллельно. В связи с этим приведенный пример проекта также не должен служить идеальным образцом, а лишь ориентиром, демонстрирующим общую логику.

Формулировка заданий к проектной работе, представленных в прил. 1, охватывает минимальное содержание первых этапов жизненного цикла ИС, которое способно дать о них общее представление и образовать в итоге оптимально обоснованный и логичный ИТ-проект, выполнимый в рамках одной дисциплины.

Таким образом, вполне допустимо, что при выполнении заданий из прил. 1 у вас, в связи со спецификой ИТ-проекта, возникнет необходимость в каких-то *обоснованных* изменениях, связанных с перечисленными выше допущениями.

2.1. Постановка проблемы и целей ИТ-проекта

Любой проект по разработке и/или внедрению новой информационной системы начинается в первую очередь с осознания со стороны заказчика наличия в организации какой-либо проблемы, которую можно решить ИТ-средствами. Далее, это осознание формулируется, превращаясь в первоначальную постановку проблемы, которая должна быть доведена до сведения всех заинтересованных лиц организации (руководство, ИТ-специалисты, работники, непосредственно связанные с рассматриваемой проблемой). Данный шаг имеет первостепенное значение, т. к. позволяет избежать приложения огромных усилий к решению несуществующей проблемы.

Один из простейших способов представления проблемы заключается в том, чтобы просто записать ее формулировку и выяснить, все ли заинтересованные лица согласны с такой постановкой (см. табл. форму ниже).

Стандартная форма постановки проблемы [28, с. 61]

Элемент	Описание
Проблема	*[формулировка проблемы]*
Воздействует на	*[указать на кого и на что оказывает влияние данная проблема]*
Результатом воздействия проблемы является	*[описание воздействия данной проблемы на заинтересованных лиц и бизнес-деятельность]*
Предполагаемое решение проблемы состоит в	*[указание предлагаемого решения]*
Выигрыш от реализации предполагаемого решения проблемы заключается в	*[список основных предоставляемых решением преимуществ]*

Неучтенное на данном этапе мнение хотя бы одного из заинтересованных лиц может полностью переориентировать формулировку проблемы и направление поиска ее решения.

В случае достижения соглашения между всеми заинтересованными лицами о необходимости решения данной проблемы она должна быть системно изучена для конкретизации, уточнения (т. е. первоначальная постановка проблемы в процессе может быть изменена) и постановки целей проекта по ее решению (т. к. решение в лоб особенно сложных плохо изученных проблем, причины и следствия которых неочевидны, может привести к ошибочной постановке целей и результатам, которые или не до конца снимают проблемную ситуацию, или совсем не удовлетворяют реальным требованиям бизнеса [31].

Определить цель проекта означает ответить на вопрос, что надо сделать для снятия проблемы. *Сформулировать цель* — значит указать направление, в котором следует двигаться, чтобы разрешить существующую проблему, показать пути, которые уводят от существующей проблемной ситуации [29, с. 79].

Любые информационные системы разрабатываются и реализуются на основе заявленных к ним требований. *Требования к информационной системе* — это описание функциональных возможностей и ограничений, накладываемых на ИС. Требования нужны для того, чтобы Разработчик (исполнитель ИТ-проекта) мог определить и согласовать с Заказчиком временные и финансовые перспективы проекта. По этой причине значительная часть требований должна быть собрана и обработана на ранних этапах создания ИС. Однако собрать на ранних стадиях все данные, необходимые для реализации ИС, удается только в исключительных случаях. На практике процесс сбора, анализа и обработки растянут во времени на протяжении всего жизненного цикла ИС, зачастую нетривиален и содержит множество подводных камней.

Бизнес-требования (business requirements) — требования, описывающие, почему организации нужна выбранная информационная система, т. е. цели, которые организация намерена достичь с ее помощью. Как правило, их высказывают те, кто финансируют проект: покупатели системы, менеджер реальных пользователей, отдел маркетинга или ответственный за концепцию продукта [30, с. 8]. Таким образом, бизнес-требование формулирует заказчик на первых этапах решения проблемы, и его формулировка совпадает с целью проекта по решению рассматриваемой проблемы, но дополнительно конкретизируется заявленными критериями к итоговому результату.

После содержательной формулировки цели и задач проекта необходимо задать *критерии* и *ограничения*, при которых будет осуществляться

их достижение. *Критерий* (от греч. *kriterion* — средство для суждения) — это признак, на основании которого производится оценка, определение или классификация чего-либо; мерило, оценка [32].

Наряду с заданными критериями большое влияние на выбор того или иного варианта решения оказывает система выделенных *ограничений* — условий, отражающих влияние внешних и внутренних факторов, которые нужно учитывать в задаче принятия решений [33]. При этом необходимо учитывать, что существуют различные источники ограничений: организационные, экономические, правовые, технические, экологические, эксплуатационные, психологические и т. д. Качественные ограничения формулируются, как правило, в терминах «не разрешается», «не допускается», а количественные — «не более», «не менее», «в интервале от... до...». Ограничения, как правило, дополняют (конкретизируют) сформулированные ранее цели и в ряде случаев могут сделать цели нереализуемыми.

После завершения исследования проблемы, формулировки цели и задач проекта проводится исследование организации (системы) в целом, в частности ее подсистемы, содержащей рассматриваемую проблему (это может быть или структурное подразделение, или конкретный(е) бизнес-процесс(ы)). Это зависит от решаемой проблемы и поставленных задач. На данном этапе рассматриваются:

- цели и задачи исследуемой системы/подсистемы;
- границы системы/подсистемы;
- состав и структура;
- выполняемые функции.

В процессе анализа обязательно участие двух сторон: Заказчика и Исполнителя системного проекта (если исполнителем будет внешнее лицо). При этом участие заказчика не ограничивается финансированием работы: от него требуется участие в анализе подсистемы, которой он управляет, формулировка цели и возможных вариантов действий.

Результатом системного анализа организации является разработка полной модели ее архитектуры, а результатом системного анализа подсистемы организации может стать или архитектура конкретного структурного подразделения (включая модели исполняемых бизнес-процессов), или только модели бизнес-процессов *As-Is*, содержащих проблему.

Вся информация о документировании и моделировании бизнес-процессов должна пройти этап рецензирования или верификации.

На этом этапе при поддержке консультантов и проектной группы проинтервьюированные сотрудники проверяют модели процессов, что гарантированно обеспечивает качество документированных моделей.

После построения и проверки перечисленных моделей проводится их экспертный анализ, направленный на выявление слабых мест в деятельности организации и определение необходимости тех или иных изменений в существующей структуре; на обеспечение поддержки бесперебойного исполнения действующих эффективных бизнес-процессов. По итогам работы формируется документ с результатами анализа, после чего рассматриваются варианты решения выявленных проблем, проводится анализ альтернатив и оценка рисков, по результатам которых выбирается наилучшее решение. Выявленное наилучшее решение требует определения ограничений и допущений на свою реализацию, а также учета требований заинтересованных лиц.

Пример выполнения задания

Начнем рассмотрение примера разработки новой информационной системы для решения проблемы детского магазина «Озорники».

Магазин «Озорники» занимается продажей детских товаров, таких как игрушки, товары для обустройства детской комнаты, товары для ухода за малышами и т. д. Продажа товаров осуществляется в розницу на площадке фирменного магазина, а также онлайн через сервис объявлений «Юла».

Миссия магазина «Озорники» — предоставление покупателям качественных детских товаров по доступным ценам.

Задачи магазина:

- привлечение клиентов;
- расширение ассортимента;
- открытие новых филиалов магазина;
- повышение конкурентоспособности.

Продукты магазина:

- различные виды детских игрушек;
- детская мебель и другие товары для обустройства детской комнаты;
- товары для активного отдыха в различные времена года (для лета — велосипеды, самокаты и проч., для зимы — санки, снегокаты и т. д.);

- спортивный инвентарь и спортивные комплексы для детей;
- товары для кормления и гигиены детей;
- разнообразные товары для обучения и развития детей;
- товары для прогулок и путешествий с детьми (коляски, автокресла и т. д.).

В марте 2021 г. магазин столкнулся с проблемой снижения объемов продаж (на 2–3 % за квартал на время начала проекта). На собрании заинтересованных лиц, представляющих руководство и отдел продаж, все согласились с существованием данной проблемы и определили, что ее решение является приоритетным, т. к. от этого зависит в целом возможность дальнейшего продолжения деятельности. В связи с чем возникла необходимость в проведении более глубокого анализа выбранной проблемы с целью выявления путей ее наилучшего решения.

Этапы анализа проблемы, ее уточнения и выявления наиболее эффективного решения здесь будут опущены, т. к. подробно рассмотрены в учебном пособии «Прикладной системный анализ в сфере ИТ: предварительное проектирование и разработка документ-концепции информационной системы» и не входят в блок заданий этого пособия, хотя рекомендуются для ознакомления с целью более подробного изучения первых этапов жизненного цикла информационных систем [31].

Представим краткие результаты этапа анализа проблемы детского магазина «Озорники» и поиска вариантов ее решения, необходимые для понимания решаемой проблемы и дальнейшей работы.

В процессе исследования проблемы (построение дерева проблем, дерева целей, моделей *As-Is* деятельности магазина), а также внешней и внутренней среды магазина, было выявлено, что при сохранении качества закупаемой и реализуемой продукции, действующей рекламной кампании, своевременном обновлении и актуализации ассортимента, а также сохранении общего спроса на детские товары и при вынужденном повышении цен на товары, появлении в городе новых конкурентов, а также развитии конкурентных преимуществ старых конкурентов снижается посещаемость магазина «Озорники» и, соответственно, падают объемы продаж.

Анализ конкурентных преимуществ показал, что все новые конкуренты, как и магазин «Озорники», используют для онлайн продаж и представления ассортимента сервисы объявлений «Юла» и «Авито», а также страницы в *Instagram* (деятельность компании *Meta Platforms Inc.* по реализации соцсетей *Instagram* и *Facebook* запрещена на территории

РФ — *прим. ред.*), часть старых конкурентов, расширив ассортимент и повысив цены, запустили для онлайн-продаж сайты, один из которых эффективно продвинулся на первую страницу поиска и стал виден большинству клиентов.

В результате анализа полученных данных и альтернативных вариантов решения проблемы руководством магазина «Озорники» было принято решение развития конкурентных преимуществ путем разработки и продвижения интернет-магазина.

Таким образом была заполнена табл. Постановка проблемы ИТ-проекта (см. с. 53).

Завершение постановки проблемы ИТ-проекта

Элемент	Описание
Проблема	Падение объемов продаж из-за снижения посещаемости магазина на фоне развития стратегии конкурентов
Воздействует на	Выручка магазина и все ее сотрудники
Результат воздействия проблемы является	Снижение прибыли, что, в свою очередь, затрудняет осуществление деятельности магазина, исполнение денежных обязательств и может привести к закрытию
Предполагаемое решение проблемы состоит в	Разработка, внедрение и продвижение интернет-магазина
Реализация предполагаемого решения проблемы с выигрышем	При реализации решения выигрыш заключается: 1) в остановке падения объемов продаж магазина в первый квартал после внедрения решения и обеспечении дальнейшего роста на 10–15 % в годовом исчислении за счет обеспечения развития конкурентных преимуществ и маркетинговой деятельности; 2) снижении издержек за счет автоматизации процесса продаж магазина

В результате был запущен ИТ-проект по разработке нового интернет-магазина «Озорники», для которого в первую очередь необходимо определить критерии и ограничения, влияющие на разработку (см. табл. форму ниже).

Ограничения, налагаемые на разработку интернет-магазина

Источник	Ограничение
Экономический	Бюджет проекта до 700 тыс. руб., без привлечения кредитных средств. Интернет-магазин должен остановить падение объемов продаж магазина «Озорники» в первый квартал после внедрения решения и обеспечить дальнейший рост на 10–15 % в годовом исчислении
Политический	Обсуждение и принятие всех проектных решений должно происходить при участии руководства магазина
Технический	Выбор технологии верстки сайта должен быть ориентирован на финансовые и временные ограничения проекта
Системный	Решение должно быть совместимо и интегрировано с существующими в магазине системами
Эксплуатационный	Сайт должен быть эргономичным и простым, не создавать помех пользователю в поисках нужной информации
График и ресурсы	В процессе выбора решения допускается привлечение сторонних специалистов, если необходимость этого обоснована. Сроки проекта реализации решения должны быть по возможности сокращены, но не должны превышать 6 месяцев

По результатам проделанной работы стало возможно сформулировать цель и бизнес-требование к ИТ-проекту. *Цель проекта* — разработка и внедрение интернет-магазина. *Бизнес-требования* — внедрение интернет-магазина должно: а) остановить падение объемов продаж в первый квартал после внедрения решения и обеспечить дальнейший рост на 10–15 % в годовом исчислении за счет обеспечения развития конкурентных преимуществ и маркетинговой деятельности; б) обеспечить снижение издержек за счет автоматизации процесса продаж магазина.

Далее, были построены модели бизнес-процессов *To-Be*, отражающие особенности работы после внедрения интернет-магазина, а также с учетом удовлетворения бизнес-требований ИТ-проекта и достижения его цели (модели бизнес-процессов не приведены, но требуются для сдачи учебной работы).

2.2. Экономическое обоснование проекта разработки и внедрения ИС

Разработка и дальнейшее сопровождение ИС требует финансовых вложений, которые для крупных систем оцениваются значительными объемами. Таким образом, возникает вопрос окупаемости новой системы, что обусловливает необходимость обоснования не только целесообразности разработки, что было описано ранее, но и экономической эффективности создания ИС, оценки ожидаемых результатов от ее функционирования и затрат, требуемых для ее разработки, ввода в действие и поддержки.

Полученные результаты отражаются в таких документах, как «ИТ-стратегия», «Технико-экономическое обоснование создания ИС», или могут содержаться в специальных разделах «Отчета об обследовании», «Документе-концепции ИС», приложении к «Техническому заданию на создание ИС».

Поскольку вопрос финансовых затрат поднимается на всей протяженности ЖЦ ИС, на каждом его этапе, то рассчитывать экономическое обоснование имеет смысл только на основе жизненного цикла разрабатываемой ИС, иначе обоснование будет неполным.

В данной главе рассматривается расчет экономии только от автоматизации соответствующего бизнес-процесса, которая сравнивается с затратами на тот же бизнес-процесс до автоматизации.

Для определения ЖЦ ИС удобно заполнить табл. 1, содержащую все этапы ЖЦ ИС каскадной модели, где:

1) *надсистема* — организация (заказчик), для которой разрабатывается информационная система;
2) *информационная система* — непосредственно разрабатываемая информационная система;
3) *подсистема* — компоненты, из которых состоит информационная система (например, модуль отчетов, текстовый модуль, интерфейс).

Каждый из перечисленных выше элементов необходимо рассмотреть в разрезе этапов жизненного цикла.

Указание длительности этапов, сроков начала и окончания превращают данную таблицу в план проекта. Очевидно, что перечень этапов может не совпадать с календарными месяцами: один этап не обязательно равен одному месяцу работы. Декомпозиция полного ЖЦ

ИС на отдельные этапы в рамках одной каскадной модели тоже может быть разной (4, 5, 6 и более этапов), однако охватывают они один и тот же перечень работ.

Таблица 1

Пример жизненного цикла ИС в каскадной модели

Характеристика этапа	Анализ требований	Проектирование	Разработка	Тестирование/ отладка	Эксплуатация/ сопровождение	Модернизация/ вывод из эксплуатации
Дата начала этапа						
Дата окончания этапа						
Надсистема(ы) ИС *«название»*						
Информационная система *«название»*						
Подсистема(ы) ИС *«название»*						
Итого денежные затраты:						
Итого временные затраты:						

Экономическую выгоду приносит только этап *эксплуатации* (также требующий вложений, но приносящий доход), а все предыдущие этапы требуют только вложений, временных и финансовых, для того, чтобы этой выгоды добиться. При расчете и планировании затрат по этапам необходимо ориентироваться на установленные ранее экономические и временные ограничения на проект (см. табл. на с. 59). Если затраты и сроки превысят ограничения, проект необходимо или пересмотреть, или отказаться от него.

В случае гибкой разработки по методике *Scrum* необходимо описать *Scrum*-модель разработки продукта для того, чтобы понимать, из каких этапов (спринтов) она состоит, какие процессы включает, какое количество ресурсов необходимо и чего ожидает заказчик в конечном итоге. Выделяют следующие этапы спринта: написание бэклога, планирование, разработка, тестирование, получение результата.

Для начала необходимо написать *бэклог продукта*, т. е. упорядоченный по степени важности список требований, предъявляемых к разрабатываемому продукту. Элементы этого списка называются *пользовательскими историями* (*user story*). Каждой истории соответствует уникальный индивидуальный номер. При этом заказчик обязательно указывает степень значимости той или иной истории, обозначает предварительную оценку объема работ и способ демонстрации результата. Примером пользовательской истории может быть требование менеджера: «Как менеджер я хочу видеть статус сотрудников в программе».

На этапе *планирования* определяется длительность спринта. Короткий спринт позволяет чаще выпускать работающие версии продукта, следовательно, чаще получать отзывы от клиента и вовремя выявлять возможные ошибки. Однако во время длинных спринтов уделяется больше времени на решение проблемы. В среднем один спринт длится около двух недель. Во время планирования спринта команда выбирает самые приоритетные пользовательские истории из бэклога продукта и определяет, каким образом будут решаться поставленные задачи. Истории, выбранные для реализации в течение данного спринта, составляют *бэклог спринта* (*sprint backlog*). Надо понимать, что каждая история должна быть реализована к концу спринта.

Далее, на этапе *разработки* происходит непосредственная разработка программы или ее компонента.

Важное место в *Scrum* занимает процесс тестирования. Существуют разные способы свести к минимуму затраты на данном этапе: от уменьшения количества историй в спринте и, как результат, снижения количества ошибок до включения тестировщиков в скрам-команду.

По окончании спринта обязательно должен получиться результат (продукт): программа, компонент программы, информационная база и проч. Команда составляет отчет, в котором показывает цели спринта, поставленные задачи и результат. На основе этого заказчик принимает решение о дальнейших действиях команды.

Далее проводится ретроспектива, основанная на решении заказчика и его обратной связи. Ее основная цель — определить, как можно улучшить процесс разработки на следующем спринте, чтобы избежать возникших проблем и повысить эффективность работы. Затем команда приступает к планированию следующего спринта.

После описания жизненного цикла ИС необходимо провести экономическое обоснование ее внедрения. Для этого необходимо рассчи-

тать полные затраты на всех этапах ЖЦ ИС и эффект от ее внедрения на этапе эксплуатации. Эффект от внедрения рассчитывается как разница между затратами на выполнение одного и того же количества задач при реализации того же бизнес-процесса в вариантах до и после внедрения ИС. На остальных этапах ЖЦ ИС затраты рассчитываются путем суммирования затрат за все выполненные виды работ. Чтобы понять, когда экономия/эффект от внедрения новой ИС превысит полные затраты на нее, строится временная зависимость полных затрат на ИС и экономии от ее использования нарастающим итогом.

Величина экономии/эффекта от использования ИС зависит от количества выполняемых задач, поэтому вначале необходимо рассчитать экономию от выполнения одной задачи, а потом умножить ее на количество задач.

Ориентируясь на построенные ранее модели бизнес-процессов *As-Is* и *To-Be*, определяются затраты на выполнение одной задачи на этапе эксплуатации до и после внедрения новой ИС (табл. 2).

Таблица 2

Условия до и после внедрения ИС

До внедрения ИС		После внедрения ИС	
Время на одну задачу (ч)		Время на одну задачу (ч)	
Стоимость обеспечения исполнения одной задачи (руб./мес. и/или руб./задача)		Стоимость обеспечения исполнения одной задачи (руб./мес. и/или руб./задача)	
и т. д.		и т. д.	

Как правило, ИС сокращает затраты времени работников на выполнение одной задачи, однако требует дополнительных постоянных затрат для своего поддержания. По этой причине важно рассчитать *точку равновесия* — количество задач (за определенный отчетный период: день, неделю, месяц или год), при котором затраты на их выполнение в режиме до и после внедрения ИС будут одинаковы. При большем количестве выполняемых задач эксплуатация ИС приводит к экономии, при меньшем — к убыткам. Очевидно, что если планируемое количество задач меньше его значения в точке равновесия, то вне-

дрение ИС становится нецелесообразным (если, конечно, отсутствуют другие положительные эффекты от внедрения).

Количество выполняемых ИС задач за каждый отчетный период после внедрения ИС задается надсистемой. На основании данных, использованных для расчета точки равновесия, легко определить убытки/экономию от внедрения ИС за каждый отчетный период на этапе эксплуатации.

В данном пособии и выполняемых на его основе заданиях вопросы налогообложения и дисконтирования денежного потока не рассматриваются.

Пример выполнения задания

Продолжим рассмотрение примера решения проблем магазина «Озорники». Пример полной модели жизненного цикла разработки и внедрения интернет-магазина «Озорники» в табличной форме по каскадной методологии представлен в табл. 3.

В данном проекте было принято решение использовать специализированный сервис для создания интернет-магазинов *Opencart*, на котором будет выбран готовый макет интернет-магазина и закуплены дополнительные платные модули, такие как модуль для системы платежей, модуль доставок СДЭК, модули для SEO-оптимизации, модули для удобного отображения товаров и их категорий. База данных также является готовым модулем в *Opencart* и будет добавлена в систему веб-разработчиком. Далее, веб-интерфейс программы будет заполнен контентом — различными товарами магазина.

Для разработки будет привлечен веб-разработчик, для наполнения сайта (300 товаров на начало внедрения) — администратор. По предварительной оценке, разработка системы займет около 1,5 месяцев (42 дня).

На этапе *эксплуатации* интернет-магазин будет активно функционировать, для чего потребуется техподдержка, осуществляемая разработчиком, актуализация содержимого (товары, информация, предложения и т. д.) администратором сайта, а также оплата подписки на использование платформы *Opencart* (хостинг и домен).

Пример представления полной модели ЖЦ разработки интернет-магазина по методологии *Scrum* представлен в табл. 4.

Таблица 3

Пример представления полной модели ЖЦ интернет-магазина «Озорники»

Этапы ЖЦ	Сбор информации	Разработка ТЗ	Разработка	Тестирование	Внедрение	Эксплуатация	Вывод из эксплуатации
Надсистема (магазин «Озорники»)	Определение ключевых задач	Разработка и согласование ТЗ; планирование финансирования этапов	Обеспечение разработки на платформе *Opencart*	Размещение сайта на тестовом сервере	Поиск домена и хостинга	Автоматизация с помощью системы бизнес-процессов магазина	Реорганизация магазина или прекращение работы
Система (интернет-магазин)	Изучение ассортимента и номенклатуры товаров, изучение клиентов магазина, изучение транзакций, действующих в магазине систем платежей и доставок	Согласование ТЗ с заказчиком и подготовка к подключению системы	Подключение модуля для управления платежами, подключение модуля для управления доставками, модуля личного кабинета и др. модулей	Ручное и автоматическое тестирование	Предоставление сотрудникам доступа к системе	Обучение и работа с системой сотрудников магазина	Приостановление работы системы
Подсистема (система управления сайтом — внутренняя логика работы)	Изучение существующих взаимодействий клиента с магазином при заказе товара	Разработка и согласование ТЗ с заказчиком и подготовка к подключению системы	Обеспечение взаимодействия модулей и БД с интерфейсом, определение функций элементов интерфейса (гиперссылки, расчет скидок)	Ручное и автоматическое тестирование	Предоставление сотрудникам доступа к системе	Обучение и работа с системой сотрудников магазина	Приостановление работы системы

Окончание табл. 3

Этапы ЖЦ	Сбор информации	Разработка ТЗ	Разработка	Тестирование	Внедрение	Эксплуатация	Вывод из эксплуатации
Подсистема (база данных)	Выявление требований к БД	Разработка ТЗ для БД	Подключение модуля БД, подключение модуля SEO-оптимизации	Тестирование БД	Предоставление сотрудникам доступа к сайту и БД	Работа сотрудников с системой	Безопасное извлечение данных из БД
Подсистема (веб-интерфейс)	Пожелания заказчика о внешнем виде интерфейса	Разработка ТЗ для веб-интерфейса, согласование с заказчиком	Разработка веб-интерфейса, наполнение контентом (товарами)	Тестирование веб-интерфейса	–	Обновление отдельных частей веб-интерфейса при необходимости, дальнейшее наполнение контентом	–
Итого денежные затраты:	10 000 руб.	20 000 руб.	117 500 руб.	15 000 руб.	10 200 руб.	Планово	20 000 руб.
Итого временные затраты:	14 дней	14 дней	42 дня	14 дней	84 дня	7 лет	1 месяц

Таблица 4

Пример представления полной модели ЖЦ разработки интернет-магазина по методологии *Scrum*

Этапы ЖЦ	Спринт 1. Разработка веб-интерфейса сайта	Спринт 2. Разработка функционала сайта	Спринт 3. Заполнение программы контентом, подключение основных модулей к сайту	Спринт 4. Размещение системы в Сети
Надсистема (магазин «Озорники»)	Формирование бюджета на разработку сайта. Согласование работ по созданию интерфейса с разработчиком.	Определение и формирование бюджета на разработку. Согласование работ по созданию сайта с разработчиком.	Определение и формирование бюджета на подключение БД. Предоставление разработчику контента сайта — изображений товаров	Определение и формирование бюджета на тестирование. Согласование работ по тестированию сайта и его основных модулей. Поиск домена и хостинга
Система (интернет-магазин)	Изучение ассортимента и номенклатуры товаров. Изучение клиентов магазина. Изучение транзакций	Изучение действующей в магазине системы платежей. Выбор модулей связи системы платежей с сайтом. Изучение действующей в магазине системы доставок. Выбор модулей связи системы доставок с сайтом. Выбор модуля личного кабинета	Подключение и согласование работы модуля системы платежей, доставок, личного кабинета и др. Предоставление сотрудникам доступа к системе	Анализ требований к системе. Тестирование системы. Исправление системы
Подсистема (система управления сайтом — внутренняя логика работы)	Изучение взаимодействий клиента с магазином при заказе товара. Определение работы кнопок и гиперссылок	Разработка расчета скидок. Разработка функционала страниц личного кабинета при оформлении заказа товара клиентом, личного кабинета сотрудника	Проверка и отладка работы клиентского взаимодействия с сайтом. Проверка и отладка работы взаимодействия сотрудников с сайтом. Предоставление сотрудникам доступа к системе	Анализ требований к системе. Тестирование системы. Исправление системы

Окончание табл. 4

Этапы ЖЦ	Спринт 1. Разработка веб-интерфейса сайта	Спринт 2. Разработка функционала сайта	Спринт 3. Заполнение программы контентом, подключение основных модулей к сайту	Спринт 4. Размещение системы в Сети
Подсистема (база данных)	–	Выбор модуля баз данных. Изучение модулей SEO-оптимизации	Подключение БД к системе. Подключение модуля SEO-оптимизации. Предоставление сотрудникам доступа к системе	Анализ требований к модулю БД и интерфейсу. Тестирование БД. Исправление БД
Подсистема (веб-интерфейс)	Определение основных страниц сайта. Определение количества и названий категорий сайта. Выбор цветов, шрифтов, логотипа. Верстка макета. Создание макета и дизайна кнопок	Разработка и добавление основных категорий сайта. Разработка страниц для оформления заказа товара клиентом	Заполнение данными — товарами магазина. Добавление баннеров и рекламных макетов. Предоставление сотрудникам доступа к интерфейсу	Анализ требований к интерфейсу. Тестирование интерфейса. Исправление интерфейса
Backlog	*Предоставить интерактивный прототип — работающую модель сайта* Обозначить структуру сайта. Разработать макет сайта. Разработать дизайн интерфейса	*Предоставить функционирующий сайт магазина с веб-интерфейсом* Разработка алгоритма работы сайта. Разработка основного функционала — опции заказа товара на сайте. Разработка основных категорий и возможности их просмотра	*Предоставить готовую программу, подключенную к базе данных* Подключение модулей SEO-оптимизации, баз данных, платежей и доставок. Заполнение базы данных. Заполнение веб-сайта контентом, баннерами и рекламными макетами	*Предоставить протестированный и отлаженный сайт в сети Интернет, который соответствует изначально поставленным требованиям и готов для внедрения* Анализ системы и требований к ней. Подготовка тестовой документации. Ручное и автоматическое тестирование. Исправление ошибок

Фазы спринта 1 представлены в табл. 5.

Таблица 5

Пример представления фаз спринта 1

Спринт 1. Разработка веб-интерфейса сайта				
Backlog	Анализ	Разработка	Тестирование	Выполнение
Предоставить работающую модель сайта. Обозначить структуру сайта. Разработать макет сайта. Разработать дизайн интерфейса	Выделить необходимые основные веб-страницы сайта, количество и названия категорий сайта. Изучить ассортимент и номенклатуру товаров, изучить клиентов, транзакции, возможности взаимодействия клиента с сайтом	Разработать макет. Для этого выбирать один из готовых макетов на платформе *Opencart*. На основе готового макета выбирать цвета, шрифты, логотипы. Также разрабатываются основные кнопки, гиперссылки для перехода	Проверка работы интерфейса, кнопок и гиперссылок, согласование их необходимости и полезности для работы сайта	Выполнено

Далее, были рассчитаны затраты на каждый этап жизненного цикла каскадной модели. В табл. 6 представлены затраты на запуск интернет-магазина в эксплуатацию.

Таблица 6

Расчет затрат на запуск интернет-магазина по этапам ЖЦ

Этапы	Виды затрат	Стоимость	Период	Итого
1. Сбор информации	Определение ключевых задач	2500 руб. (оплата работы разработчика)	14 дней	10 000 руб.
	Разработка концепции интерфейса сайта, изучение существующих взаимодействий клиента с магазином при заказе товара	2500 руб. (оплата работы разработчика)		
	Сбор информации о действующих системах платежей, доставок, изучение ассортимента и номенклатуры товаров, изучение клиентов магазина, изучение транзакций.	2500 руб. (оплата работы разработчика)		
	Выявление требований к БД	2500 руб. (оплата работы разработчика)		

Продолжение табл. 6

Этапы	Виды затрат	Стоимость	Период	Итого
Разработка ТЗ проекта	Разработка и согласование ТЗ. Планирование финансирования этапов	2000 руб. (з/п разработчику за выполненную работу) 2000 руб. (з/п менеджеру за выполненную работу) *Итого:* 4000 руб.	14 дней	20000 руб.
	Согласование ТЗ с заказчиком и подготовка к подключению системы платежей, доставок (выбор наиболее подходящих модуля)	4000 руб. (з/п разработчику за выполненную работу)		
	Согласование ТЗ с заказчиком и подготовка к подключению системы управления сайтом (выбор наиболее подходящих модулей)	3000 руб. (з/п разработчику за выполненную работу)		
	Разработка ТЗ для БД (выбор наиболее подходящего модуля БД)	4000 руб. (з/п разработчику за выполненную работу)		
	Разработка ТЗ для веб-интерфейса, согласование с заказчиком (выбор основных страниц, дизайна, шрифтов для сайта)	5000 руб. (з/п разработчику за выполненную работу)		
Разработка веб-сайта	Обеспечение разработки на платформе *Opencart* (покупка шаблона магазина)	500 руб. (покупка шаблона) 10000 руб. (з/п разработчику за выполненную работу)	42 дня	117500 руб.
	Подключение модуля для управления платежами, доставками и др. (68 дополнительных модулей к веб-сайту)	15000 руб. (з/п разработчику за выполненную работу) 34000 руб. (покупка модулей, со средней стоимостью модуля 500 руб.)		

Продолжение табл. 6

Этапы	Виды затрат	Стоимость	Период	Итого
Разработка веб-сайта	Подключение модулей для управления сайтом (3 дополнительных модуля к веб-сайту для настройки панели управления для администратора, менеджера и личный кабинет для клиента)	4000 руб. (з/п разработчику за выполненную работу) 3000 руб. (покупка модулей, со средней стоимостью модуля 1000 руб.)	42 дня	117500 руб.
	Подключение модуля БД, подключение модуля SEO-оптимизации (18 дополнительных модулей к веб-сайту)	6000 руб. (з/п разработчику за выполненную работу) 9000 руб. (покупка модулей со средней стоимостью модуля 500 руб.)		
	Разработка веб-интерфейса. Наполнение контентом (товарами). (Создание категорий и товаров (300 шт.) сайта, подключение 50 дополнительных модулей к веб-интерфейсу)	5000 руб. (з/п разработчику за выполненную работу) 25000 руб. (покупка модулей со средней стоимостью модуля 500 руб.) 6000 руб. (з/п администратора сайта за 300 товаров)		
Тестирование	Размещение сайта на тестовом сервере	3000 руб. (з/п тестировщику за выполненную работу)	14 дней	15000 руб.
	Ручное и автоматическое тестирование системы управления платежами, доставками	3000 руб. (з/п тестировщику за выполненную работу)		
	Ручное и автоматическое тестирование системы управления сайтом	3000 руб. (з/п тестировщику за выполненную работу)		
	Тестирование БД	3000 руб. (з/п тестировщику за выполненную работу)		
	Тестирование веб-интерфейса	3000 руб. (з/п тестировщику за выполненную работу)		

Окончание табл. 6

Этапы	Виды затрат	Стоимость	Период	Итого
Внедрение	Домен и хостинг	350 руб. (домен за 2 мес.) 350 руб. (хостинг за 2 мес.)	84 дня	10700 руб.
	Предоставление сотрудникам доступа к системе платежей и доставок.	10000 руб. (з/п разработчику за обучение сотрудников работе с системой)		
	Предоставление сотрудникам доступа к системе управления сайтом			
	Предоставление сотрудникам доступа к сайту и БД			
Итого:				**173 200 руб.**

Вывод из эксплуатации должен быть учтен в расходах компании, они определяются предприятием, но т. к. предполагается, что интернет-магазин будет использоваться в течение длительного срока (7 лет), невозможно заранее спрогнозировать точные виды работ и сосчитать точные затраты на этот этап, поэтому в данном примере этот вид затрат не будет учтен.

Издержки на выполнение автоматизируемого бизнес-процесса до внедрения интернет-магазина

Рассмотрим издержки до внедрения интернет-магазина (*AS-IS*). Когда товар куплен у поставщика и размещен в магазине, нужно оповестить покупателей о наличии нового товара в магазине для их привлечения. До внедрения веб-сайта для данной цели был использован такой сервис объявлений, как «Юла». Стоимость размещения одного объявления на «Юле» равна 16 руб., заработная плата администратору за размещение одного объявления на «Юле» составляет 30 руб., определяется надсистемой — работодателем. Итого 46 руб. за размещение информации об одном товаре.

Для того чтобы оформить доставку и покупки товара, необходимо созвониться с покупателем. Время, необходимое для разговора с покупателем, составляет 25 мин, стоимость такого звонка составляет 41,25 руб. (были рассмотрены тарифы МТС — 1,65 руб./мин).

$$\text{Стоимость звонка} = 1{,}65 \text{ руб.} \times 25 \text{ мин} = 41{,}25 \text{ руб.}$$

Считая, что все свое рабочее время менеджер тратит на созвоны с покупателями по вопросам исполнения их заказов, рассчитаем количество заказов, которые менеджер должен обслужить за месяц. Его полный рабочий день составляет 8 ч, работая 5 дней в неделю в течение месяца (четырех недель), он выполняет заказы в течение 8 × 5 × 4 = 160 ч или 160 × 60 = 9600 мин. Поскольку один заказ требует на свое выполнение 25 мин, то количество заказов, которое менеджер выполнит за месяц, составит 9600 : 25 = 384. Соответственно, стоимость выполнения одного заказа составит 25 000 : 384 = = 65,1 руб.

Тогда с учетом расходов на телефонную связь в размере 41,25 руб./заказ затраты на звонок по одному заказу составят 41,25 + 65,1 = 106,35 руб. С учетом стоимости размещения информации о товаре на сервисе «Юла» переменные затраты на процесс продаж составят 46 + 106,35 = = 152,35 руб./заказ.

В качестве постоянных затрат на бизнес-процесс в *AS-IS* рассматриваем те затраты, которые не зависят от количества оформленных в магазине заказов товаров.

Постоянные затраты на бизнес-процесс состоят из оплаты интернета МТС по тарифу 550 руб./месяц. Также для более эффективной продажи товаров покупается аккаунт «Расширенный магазин» на «Юле» стоимостью 1864 руб./месяц. Кроме того, оплачивается реклама аккаунта на Юле в социальных сетях за 2500 руб./месяц. Итого 4914 руб.

Издержки на выполнение автоматизируемого бизнес-процесса после внедрения интернет-магазина

Теперь рассмотрим издержки на этап эксплуатации после внедрения (*To-Be*). Оповещение покупателей о наличии товаров теперь осуществляется через веб-сайт путем добавления товаров в интернет-магазин. Заработная плата администратору за размещение информации об одном товаре составляет 7 руб. и определяется надсистемой — работодателем. Заработная плата за размещение на сайте определена меньше, чем за размещение на Юле, т. к. администратор тратит меньше усилий на эту работу. Итого 7 руб.

В реализации бизнес-процесса *To-Be* созвоны с покупателями отсутствуют, поэтому нет расходов на телефонную связь, покупатель сам формирует заказ, а менеджеру остается уточнить корректность заказа

и выбранной формы доставки по электронной почте. На такое уточнение он тратит всего 10 мин, поэтому при условии полной загрузки этой работой ему надо обслужить за месяц 9600/10 = 960 заказов. Тогда стоимость обслуживания одного заказа при его зарплате в 25 000 руб./мес. составит 25 000 : 960 = 26,31 руб./заказ.

Магазин торгует в розницу, а товар закупает очень маленькими партиями (причем в одной партии представлены товары разных размеров). Один и тот же товар, но разного размера, представлен разными карточками товара в ИС склада магазина, а на сайте — разными страницами. По этой причине каждому сделанному заказу предшествует размещение заказываемого экземпляра товара в интернет-магазине (как и при размещении на Юле). Соответственно, переменные издержки будут складываться из стоимости размещения информации об одном экземпляре товара на сайте и стоимости обслуживания заказа: 7 + 26,31 = 33,31 руб./заказ. Это и есть значение переменных издержек на один заказ в данном бизнес-процессе (другие бизнес-процессы магазина тоже вносят свой вклад как в переменные, так и постоянные издержки всего предприятия, но они не связаны с использованием ИС).

В варианте *To-Be* затраты времени менеджера на администрирование одного заказа сокращаются в 2,5 раза. Даже если он до внедрения ИС был занят этим полностью, то очевидно, что сейчас у него освобождается рабочее время (ниже показан план продаж — планируемое количество заказов в месяц находится на уровне 700), поэтому работодатель использует освободившееся рабочее время менеджера для выполнения других видов работ. При этом корректность нашего расчета стоимости обслуживания одного заказа в 26,31 руб. сохраняется.

В качестве постоянных затрат на ИС в процессе *To-Be* рассматриваем те затраты, которые не зависят от количества оформленных в магазине заказов товаров.

Постоянные затраты на веб-сайт состоят из оплаты подписки на сайт в размере 350 руб./месяц, оплаты Интернета МТС по тарифу 550 руб./месяц, рекламы сайта 2000 руб./месяц, SEO-продвижение 2500 руб./месяц. Также необходимо, чтобы разработчик сайта осуществлял его тех. поддержку. За это ему платится 20 000 руб./месяц. Итого 25 400 руб (табл. 7).

Таблица 7

Временные и денежные затраты по бизнес-процессам

Бизнес-процесс		До внедрения ИС (*As-Is*)		После внедрения ИС (*To-Be*)	
		Временные затраты	Денежные затраты	Временные затраты	Денежные затраты
Переменные затраты/заказ					
1	Размещение товара в интернет-магазине	–	46 руб.	–	7 руб.
2	Оформление доставки и покупки товара	25 мин	106,35 руб.	10 мин	26,31 руб.
Итого:			152,35 руб./заказ		33,31 руб./заказ
Постоянные издержки					
1	Интернет	1 мес.	550 руб.	1 мес.	550 руб.
2	Техподдержка		0 руб.		20000 руб.
3	Затраты на использование сервиса/сайта		1864 руб.		350 руб.
4	Реклама		2500 руб.		2000 руб.
5	SEO-продвижение		0 руб.		2500 руб.
Итого:		**1 мес.**	**4914 руб./мес.**	**1 мес.**	**25400 руб./мес.**

Как можно увидеть из табл. 7, с внедрением интернет-магазина происходит заметная экономия времени и денежных затрат на 1 оформленный в магазине заказ товара.

Таким образом, формула точки равновесия:

Пусть x — точка равновесия

152,35 — переменные затраты на один заказ *As-Is*

4914 — постоянные затраты *As-Is*

25400 — постоянные затраты *To-Be*

33,31 — переменные затраты на один заказ *To-Be*

$152{,}35x + 4914$ — полные издержки *As-Is*

$25400 + 33{,}31x$ — полные издержки *To-Be*

Считаем точку равновесия:

Полные издержки *As-Is* = Полные издержки *To-Be*

$152{,}35x + 4914 = 25400 + 33{,}31x$

$119{,}04x = 20486$

$X = 172{,}09 = 173$ заказов/мес. (поскольку количество заказов может быть только целым числом, округляем в большую сторону). При

большем количестве заказов в месяц интернет-магазин будет обеспечивать экономию, при меньшем — приносить убытки.

Рассмотрим затраты на бизнес-процесс продаж в зависимости от количества выполненных заказов до внедрения ИС (рис. 17).

Постоянные издержки: прямая линия $y = 4914$.

Переменные издержки: $y = 152{,}35x$.

Полные издержки =
= Переменные издержки + Постоянные издержки

$$y = 152{,}35x + 4914$$

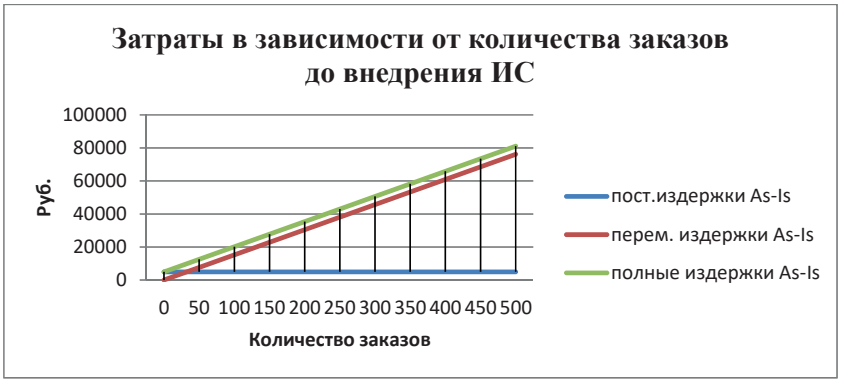


Рис. 17. Затраты в зависимости от количества выполненных заказов до внедрения ИС

Рассмотрим затраты на тот же бизнес-процесс от количества выполненных заказов после внедрения ИС (рис. 18).

Постоянные издержки: прямая линия $y = 25400$.

Переменные издержки: $y = 33{,}31x$.

Полные издержки =
= Переменные издержки + Постоянные издержки

$$y = 33{,}31x + 25400$$

Сравним полные затраты *As-Is* и *To-Be* в зависимости от объема выпуска (рис. 19).

Полные издержки *As-Is*: $y = 152{,}35x + 4914$.

Полные издержки *To-Be*: $y = 25400 + 33{,}31x$.

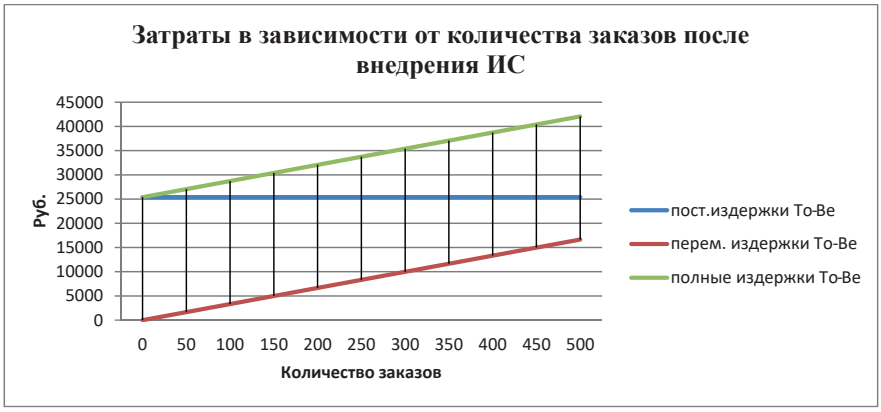


Рис. 18. Затраты в зависимости от количества выполненных заказов после внедрения ИС

Точка пересечения графиков полных издержек *As-Is* и *To-Be* — 172,09, поэтому точка равновесия равна 173 заказов/мес.

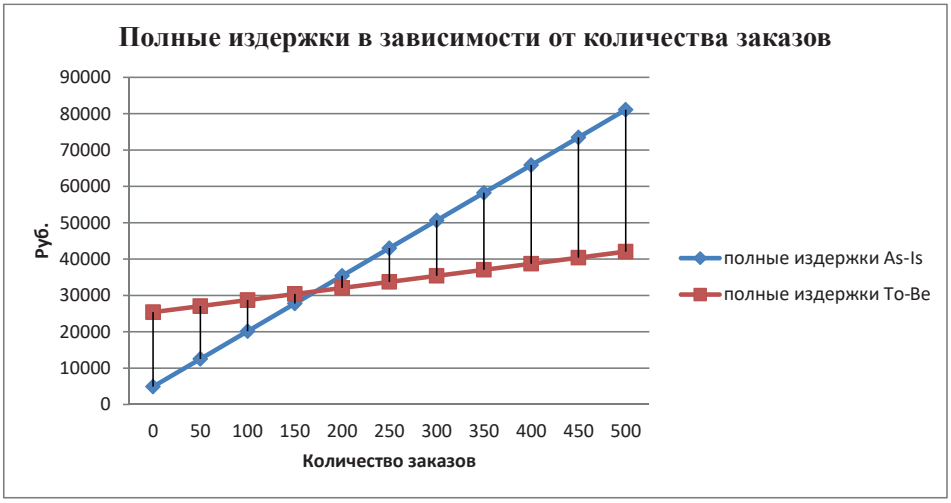


Рис. 19. Полные издержки в зависимости от количества заказов

На рис. 20 представлено планируемое количество обработанных ИС заказов в месяц с момента запуска проекта (за весь ЖЦ системы). Такое количество заказов определено руководством магазина согласно маркетинговому плану.

Рис. 20. Количество обработанных ИС заказов в месяц с момента запуска проекта (за весь ЖЦ системы)

Ниже представлен график затрат на прием заявок в месяц *As-Is* и *To-Be* (рис. 21) согласно плану выполнения заказов/продаж (рис. 20), разница между которыми является экономией/потерями от внедрения ИС на этапе эксплуатации (с 6-го месяца).

Затраты *As-Is*/мес. = 152,35 × Число заказов + 4914 (пост. затраты)

Затраты *To-Be*/мес. = 33,31 × Число заказов + 25400 (пост. затраты)

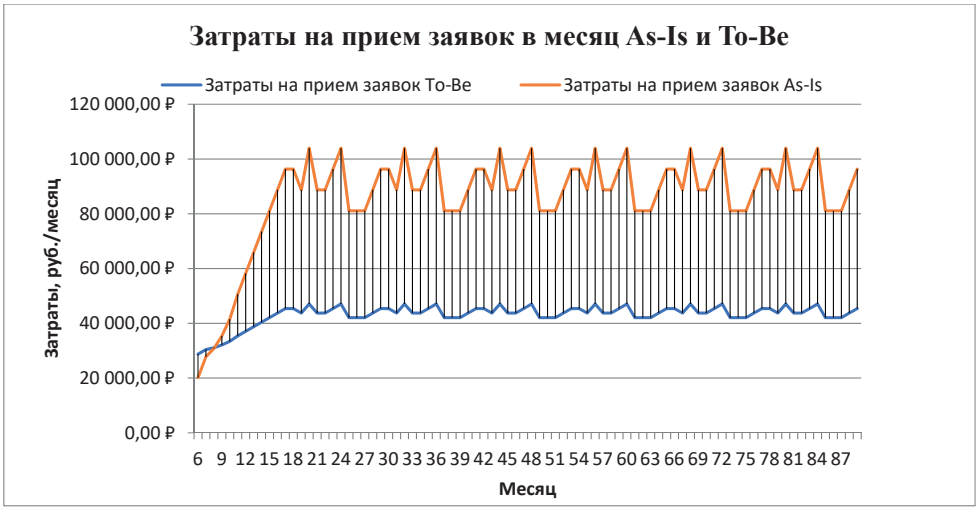


Рис. 21. График затрат на прием заявок в месяц *As-Is* и *To-Be* с момента начала эксплуатации ИС (6 месяц)

Ниже представлен график затрат на прием заявок в месяц *As-Is* на этапе эксплуатации и *To-Be* на протяжении всего ЖЦ (рис. 22). Внедрение веб-сайта начинается на 4 месяце, а использование на 6 месяце. Экономия зависит от количества обрабатываемых заявок в месяц (рис. 20).

Затраты *As-Is*/мес. = 152,35 × Число заказов/мес. + 4914 (пост. затраты/мес.)

Затраты *To-Be*/мес. до начала эксплуатации = = Расходы на создание ИС/мес.

Затраты *To-Be*/мес. на этапе эксплуатации = = 33,31 × Число заказов/мес. + 25400 (пост. затраты/мес.)

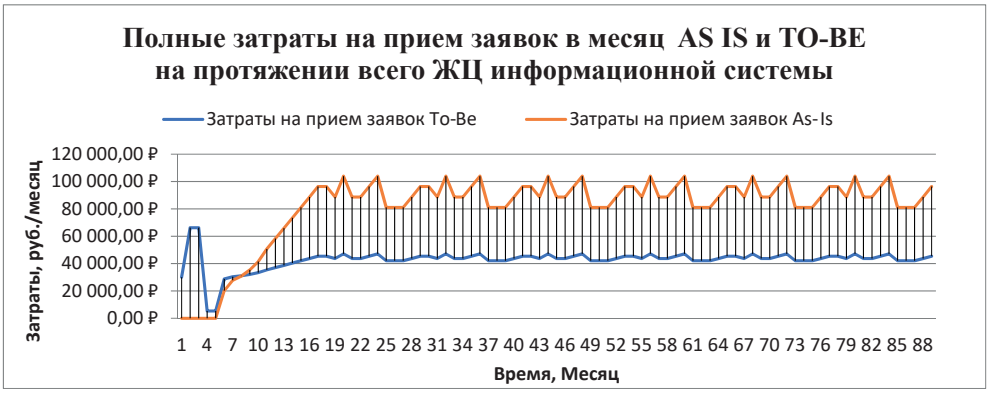


Рис. 22. График полных затрат на протяжении всего ЖЦ информационной системы (*To-Be*) и затраты на прием заявок в месяц *As-Is*

Экономический эффект от внедрения ИС помесячно представлен на графике ниже (рис. 23). Считается по следующей формуле:

Экономический эффект (в мес.) = = Затраты на обработку заявок *As-Is* — Полные затраты *To-Be*

Экономический эффект от внедрения ИС нарастающим итогом представлен на графике ниже (рис. 24). Считается по следующей формуле:

$$\text{Экономический эффект} = \sum_{i=1}^{n}\left(\text{Затраты на прием заявок } As\text{-}Is - \text{Полные затраты на ИС}\left(To\text{-}Be\right)\right)$$

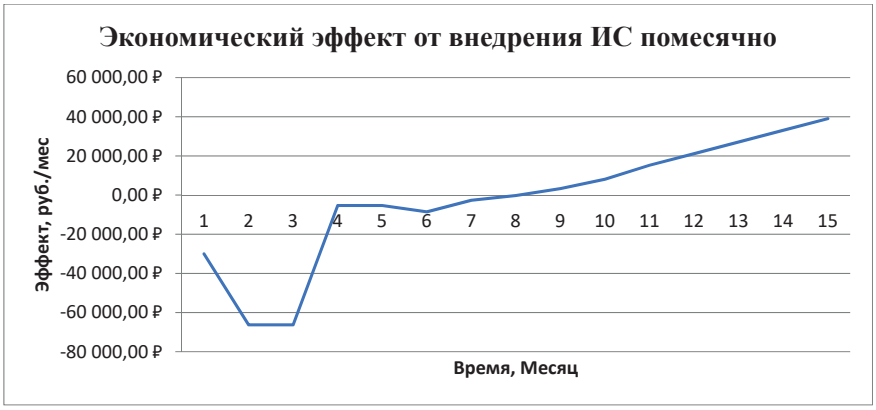


Рис. 23. Экономический эффект от внедрения ИС помесячно

Рис. 24. Экономический эффект от внедрения ИС нарастающим итогом на протяжении всего ЖЦ ИС

На графике рис. 24 видно, что интернет-магазин окупается за 16 месяцев с начала проекта. Полные затраты на проект составляют 181 339,20 руб., и это больше, чем инвестиционные затраты на создание и внедрение информационной системы = 173 200 руб., т. к. про веб-сайт клиенты узнают не сразу, и сначала интернет-магазин будет приносить убыток. Пока система работает ниже точки равновесия, ее необходимо поддерживать.

Контрольные вопросы для самоконтроля

1. Зачем нужно учитывать мнения заинтересованных лиц при постановке проблемы?
2. В чем заключается важность правильной постановки целей проекта?
3. Для чего нужны требования в ИТ-проекте?
4. Почему важно установить критерии и ограничения достижения целей проекта?
5. Чем отличается модель ЖЦ разработки ИС от модели полного ЖЦ ИС?
6. Что такое точка равновесия при эксплуатации ИС?
7. Что такое период окупаемости проекта внедрения ИС?
8. В какой момент значение полных инвестиций (расходов на создание, внедрение и начальную эксплуатацию ИС) в проект разработки и внедрения ИС достигает максимального значения?
9. В чем состоит принципиальное отличие постоянных и переменных издержек на эксплуатацию ИС?
10. В какой реализации бизнес-процесса переменные издержки на обработку одного заказа/клиента оказываются ниже — *As-Is* или *To-Be*?
11. При каких условиях внедрение ИС на предприятии или в организации оказывается невыгодным?
12. В материалах пособия расчеты постоянных и переменных издержек проводятся без учета уплаты налогов. Как изменятся зна-

чения точки равновесия и периода окупаемости проекта, если учитывать уплату налогов?

13. В материалах пособия при расчете периода окупаемости не проводится дисконтирование денежного потока (не учитывается ставка дисконтирования). Как изменится значение периода окупаемости, если такое дисконтирование учитывать?
14. Предположим, что финансирование проекта разработки и внедрения ИС проводится за счет кредитных средств. Определите основные параметры кредита (основные характеристики и их значения согласно сделанным ранее расчетам по проекту).
15. Расчет точки равновесия при эксплуатации ИС проводится по той же математической модели, что и расчет точки безубыточности для предприятия. Какой параметр используется вместо цены единицы продукции?

Список библиографических ссылок

1. ГОСТ Р 57193–2016. Системная и программная инженерия. Процессы жизненного цикла систем = Systems and Software Engineering. System life cycle processes. URL: https://docs.cntd.ru/document/1200141163 (дата обращения: 07.07.2022).
2. ГОСТ Р 50-605-80-93. Система разработки и постановки продукции на производство. Термины и определения. URL: https://files.stroyinf.ru/Data2/1/4293827/4293827526.htm (дата обращения: 07.07.2022).
3. IEEE Standard Glossary of Software Engineering Terminology // IEEE Standards Board. 1990. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=159342 (дата обращения: 07.07.2022).
4. Winston W. Royce. Managing the Development of Large Software Systems // Technical Papers of Western Electronic Show and Convention (WesCon). Los Angeles, 1970.
5. ГОСТ 34.601–90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания. URL: https://docs.cntd.ru/document/1200006921 (дата обращения: 07.07.2022).
6. ГОСТ 34.602–89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы. URL: https://docs.cntd.ru/document/1200006921 (дата обращения: 07.07.2022).

7. ГОСТ 34.201–89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначения документов при создании автоматизированных систем. URL: https://docs.cntd.ru/document/1200006924 (дата обращения: 07.07.2022).
8. Документирование по ГОСТ 34 — это просто. URL: https://habr.com/ru/post/122700/ (дата обращения: 09.09.2021).
9. Чавалах А. Разработка технического задания. Что это такое, зачем оно нужно, с чего начать и как оно должно выглядеть? URL: https://www.klerk.ru/soft/articles/333385/ (дата обращения: 09.07.2022).
10. ГОСТ Р ИСО/МЭК 12207–2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. URL: https://docs.cntd.ru/document/1200082859 (дата обращения: 07.07.2022).
11. Системы менеджмента качества. Основные положения и словарь = Quality management systems. Fundamentals and vocabulary. URL: https://www.istu.edu/docs/education/fgos_14/ISO_9000-2005rus.pdf (дата обращения: 09.07.2022).
12. Анисимов В. И. Проектирование информационных систем // Учебная и научная деятельность В. И. Анисимова. URL: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/ (дата обращения: 09.07.2022).
13. Бубнов И. Методологии разработки: Waterfall // GeekBrains. URL: https://gb.ru/posts/waterfall (дата обращения: 09.07.2022).
14. V-модель // QALight — Цетр подготовки IT-специалистов. База знаний. URL: https://qalight.ua/ru/baza-znaniy/v-model-v-model-2/ (дата обращения: 10.07.2022).
15. Методологии управления информационными проектами // Хабр. URL: https://habr.com/ru/post/244003/ (дата обращения: 09.07.2022).
16. Модели жизненного цикла программного обеспечения. URL: https://habr.com/ru/post/111674/ (дата обращения: 09.07.2022).
17. Agile-манифест разработки программного обеспечения. URL: https://agilemanifesto.org/iso/ru/manifesto.html (дата обращения: 09.07.2022).

18. Вольфсон Б. Гибкие методологии разработки. URL: https://strategium.space/wp-content/uploads/2018/07/Gibkie-metodologii.pdf (дата обращения: 11.07.2022).
19. Agile в России // ScrumTrek. URL: https://agilesurvey.ru/(дата обращения: 09.07.2022).
20. Takeuchi H., Nonaka I. The New Product Development Game // Harvard Business Review. 1986. URL: https://hbr.org/1986/01/the-new-new-product-development-game (дата обращения: 09.07.2022).
21. Березин А. Методологии разработки ПО: Kanban // GeekBrains. URL: https://gb.ru/posts/kanban_howto (дата обращения: 09.07.2022).
22. Матюшкин Е. Экстремальное программирование — мифы и реальность // Записки трезвого практика. URL: http://skipy.ru/philosophy/xp.html (дата обращения: 09.07.2022).
23. Рефакторинг: улучшение существующего кода / М. Фаулер, К. Бек, Д. Брант [и др.]. СПб. : Символ-Плюс, 2009. 432 с.
24. Березин, А. Методологии разработки ПО: Agile // GeekBrains. URL: https://gb.ru/posts/methodologies_agile (дата обращения: 09.07.2022).
25. Блок-схема выбора оптимальной методологии разработки ПО // Хабр. URL: https://habr.com/ru/post/297612/ (дата обращения: 11.07.2022).
26. Trello. URL: https://trello.com/(дата обращения: 11.07.2022).
27. Воловик П. Что такое Trello и как им пользоваться // Медиа нетологии. URL: https://netology.ru/blog/trello (дата обращения: 14.07.2022).
28. Леффингуэлл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. М. : Вильямс, 2002. 448 с.
29. Вишнякова А. Ю., Берг Д. Б. Прикладной системный анализ в сфере ИТ: предварительное проектирование и разработка документ-концепции информационной системы. Екатеринбург : Изд-во Урал. ун-та, 2020. 179 с.
30. Чернышов В. Н., Чернышов А. В. Теория систем и системный анализ. Тамбов : Изд-во ТГТУ, 2008. 96 с.

31. Вигерс К., Битти Дж. Разработка требований к программному обеспечению ; пер. с англ. 3-е изд., доп. М. : Издательство Русская редакция; СПб. : БХВ-Петербург, 2014. 736 с.
32. Голубева Т. Б. Основы моделирования и оптимизации процессов и систем сервиса. Екатеринбург : Изд-во Урал. ун-та, 2017. 108 с.
33. Катанаева М. А., Шушерина О. А., Бывшев В. И. Управление качеством образовательных услуг вуза: системный взгляд // УЭкС. 2013. № 9 (57). URL: https://cyberleninka.ru/article/n/upravlenie-kachestvom-obrazovatelnyh-uslug-vuza-sistemnyy-vzglyad (дата обращения: 08.10.2022).

Приложение 1

Практические задания

Задание 1. Постановка проблемы и целей ИТ-проекта

1. Выберите организацию для дальнейшего анализа. В качестве выбора организации лучше всего подойдет та, которая ранее уже рассматривалась в рамках изучения других дисциплин. Организация должна быть знакомой, лучше всего, если студент там работал или работает в данный момент и хорошо знаком с ее бизнес-процессами, существующими проблемами. Также можно рассмотреть организации, в которых работают родственники или знакомые, если они готовы представить, при необходимости, всю нужную информацию.
2. Подготовьте текстовое описание организации, где вкратце содержится следующая информация: сфера деятельности; миссия, цели, задачи; продукты/услуги; адрес; история и т. д. Источниками данной информации могут стать сотрудники организации, корпоративный сайт или ее нормативные документы.
3. Сформулируйте одну проблему организации, которая будет решаться в рамках работы путем разработки/внедрения ИС. Наиболее подходящие:
   - низкая производительность труда;
   - низкое качество продуктов/услуг;
   - большие затраты на управление процессами в организации, частые ошибки исполнителей (являются следствием сложившейся системы управления);

- потребность в продвижении продуктов/услуг;
- необходимость повышения качества коммуникации с клиентами.

Данные проблемы можно решить разработкой и внедрением различных информационных систем (ИС). Однако не каждое приложение разрабатывается для решения определенной проблемы, некоторые из них создаются для того, чтобы воспользоваться предоставляемыми рынком возможностями, даже если существование проблемы еще неочевидно. Таким образом, объектом исследования может стать и стартап, направленный на разработку новых ИТ-продуктов. В таком случае опишите направление деятельности старпата и все проблемы, которые призван решить новый ИТ-продукт для своих пользователей.

4. Постройте модели бизнес-процессов *As-Is*, содержащих выявленную проблему. Важным условием является иллюстрация в моделях наличия проблемы.
5. Определите всевозможные критерии и ограничения, накладываемые на проект, решающий выбранную проблему организации. Для примера некоторые возможные источники критериев и ограничений представлены в табл. форме ниже.

Возможные источники ограничений системы

Источник	Образцы вопросов
Экономический	Какие финансовые или бюджетные ограничения следует учесть? Существуют ли соображения, касающиеся себестоимости и ценообразования? Существуют ли вопросы лицензирования? Существует ли и какой экономический выигрыш должен принести проект (увеличение прибыли в/на сколько)? и т. д.
Политический	Существуют ли внешние или внутренние политические вопросы, влияющие на потенциальное решение? Существуют ли проблемы в отношениях между подразделениями? Существует ли и какой политический эффект может принести проект (снижение конфликтов в/на сколько раз?) и т. д.

Источник	Образцы вопросов
Технический	Существуют ли ограничения в выборе технологий? Должны ли мы работать в рамках существующих платформ или технологий? Запрещено ли использование любых новых технологий? Должны ли мы использовать какие-либо закупаемые пакеты программного обеспечения?
Системный	Будет ли решение создаваться для наших существующих систем? Должны ли мы обеспечивать совместимость с существующими решениями? Какие операционные системы и среды должны поддерживаться? Существует ли и какой системный эффект может принести проект (снижение ошибок между модулями системы в/на сколько раз?)
Эксплуатационный	Существуют ли ограничения информационной среды или правовые ограничения? Юридические ограничения? Требования безопасности? Какими другими стандартами мы ограничены?
График и ресурсы	Определен ли график? Какие конкретные сроки нужно соблюсти? Ограничены ли мы существующими ресурсами? Можем ли мы привлекать работников со стороны? Можем ли мы увеличить штат? Временно? Постоянно?

6. С учетом анализа моделей *As-Is*, критериев и ограничений проекта, а также с учетом возможных рисков, предложите ИТ-решение для выбранной проблемы и заполните табл. Первоначальная постановка проблемы, приведенную в подглаве 2.1 (с. 53).
7. Сформулируйте цель и бизнес-требование проекта.
8. Постройте модели бизнес-процессов *To-Be* с учетом особенностей выбранного решения, бизнес-требований и цели ИТ-проекта.

Критерии оценивания задания представлены в прил. 2.

Задание 2. Экономическое обоснование проекта

1. Составьте полную модель ЖЦ ИС в рамках традиционной методологии (см. табл. 3).
   1.1. Определите этапы полного ЖЦ анализируемой ИС (включая вывод из эксплуатации). Для этого выберите модель ЖЦ в рамках традиционной методологии. Определите надсистему (обычно — предприятие или его подразделение, или более крупная ИС) для анализируемой ИС, а также подсистемы анализируемой ИС.
   1.2. Составьте таблицу, заголовками столбцов которой являются этапы ЖЦ, а названиями строк — надсистема, сама ИС и ее подсистемы. Заполните ячейки табл. видами работ. Строка надсистемы заполняется действиями, которые осуществляются с ее стороны в отношении ИС (требования, финансирование, согласование и др.). Содержание нескольких ячеек табл. (по согласованию с преподавателем) поясните более подробно в комментариях под таблицей.
2. Составьте полную модель ЖЦ ИС в рамках гибкой методологии. Определите этапы полного ЖЦ анализируемой ИС согласно выбранной гибкой методологии разработки. Составьте табл. аналогично п. 1, заменив этапы ЖЦ на отдельные спринты. В табл. добавьте строку с бэклогом каждого спринта и проекта в целом (см. табл. 4). Как минимум для одного спринта (для методологии *Scrum*) составьте отдельную табл. с указанием конкретных работ на каждой фазе этого спринта (см. табл. 5).
3. Проведите расчет затрат на каждом этапе ЖЦ ИС (модель ЖЦ по традиционной методологии) без учета налогов, процентных ставок и др. (см. табл. 6).
   3.1. Для расчета затрат на всех этапах ЖЦ ИС, кроме этапа эксплуатации, подробно декомпозируйте работы на данном этапе, рассчитайте стоимость всех используемых ресурсов, включая затраты времени работников. Затраты на заработную плату рассчитайте с учетом квалификации работников и их трудозатрат. Одновременно определите полную продолжительность каждого этапа. Результаты занесите в табл. с моделью ЖЦ.
   3.2. Для этапа эксплуатации рассчитайте как затраты, так и величину экономического эффекта (экономию) от использования

ИС. Для этого необходимо рассчитать полные (постоянные и переменные) затраты на автоматизируемый бизнес-процесс до (*As-Is*) и после (*To-Be*) автоматизации (внедрения ИС). После этого рассчитайте точку равновесия, при которой добавившиеся постоянные затраты на содержание ИС будут покрыты экономией от снижения переменных затрат на обслуживание каждого заказа/клиента за счет использования ИС. Постройте графики функций затрат на автоматизируемый бизнес-процесс до (*As-Is*) и после (*To-Be*) автоматизации от объема обслуживаемых заказов.

4. Проведите расчет финансовых потоков по проекту на протяжении всего ЖЦ ИС в рамках выбранной модели.
   4.1. Вначале рассчитайте финансовые потоки (помесячно) на этапе эксплуатации. Для этого необходимо задать план обработки запросов/выполнения заказов (обычно — помесячно) на протяжении всего этапа эксплуатации или выбранного для анализа временного периода (от 1 до 5 лет). Используя результаты п. 3, рассчитайте, согласно заданному плану продаж, объемы полных издержек до (*As-Is*) и после (*To-Be*) автоматизации помесячно. Рассчитайте баланс (разницу) между ними. Изобразите все три рассчитанных временных ряда на одном графике.

   Затем необходимо расширить временные ряды баланса затрат (разницы) полных издержек до (*As-Is*) и после (*To-Be*) автоматизации помесячными затратами на протяжении всего ЖЦ ИС, а также баланса между ними, объединив таким образом результаты расчетов по п. 3 и 4 на одной временной оси. Допускается ограничиться только одним последним временным рядом (рядом баланса). Эти временные ряды постройте на одном графике (помесячно).
   4.2. Рассчитайте временные ряды полных издержек до (*As-Is*) и после (*To-Be*) автоматизации, а также их баланса нарастающим итогом, просуммировав все предыдущие значения помесячных рядов (допускается использовать только один ряд — ряд баланса). Постройте соответствующие ряды (ряд) на графике. По нему определите точку возврата инвестиций (окупаемости проекта), а также максимальную сумму расходов по проекту.

Критерии оценивания задания представлены в прил. 2.

Приложение 2

Критерии оценивания выполнения практических заданий

«Отлично»	Работа выполнена самостоятельно в полном объеме и в заданный срок*, оформлена с соблюдением установленных требований. Присутствуют собственные обобщения, заключения и обоснованные выводы. **Нарушение срока сдачи работы понижает максимальную оценку до уровня «хорошо».*
«Хорошо»	Работа выполнена самостоятельно и в заданный срок*, но с незначительными отклонениями от требований к содержанию и оформлению. **Нарушение срока сдачи работы понижает максимальную оценку до уровня «удовлетворительно».*
«Удовлетворительно»	Работа выполнена самостоятельно и в заданный срок*, но с незначительными отклонениями от требований к содержанию и/или с серьезными ошибками в техническом оформлении. **Нарушение срока сдачи работы понижает максимальную оценку до уровня «неудовлетворительно».*
«Неудовлетворительно»	Работа выполнена не в полном объеме, с серьезными ошибками.
«Неудовлетворительно»	Отсутствие выполненной работы или работа выполнена несамостоятельно.

Учебное издание

Берг Дмитрий Борисович
Зверева Ольга Михайловна
Вишнякова Алина Юрьевна

УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ ИНФОРМАЦИОННЫХ СИСТЕМ

Редактор *Н. Ф. Тофан*
Верстка *Е. В. Ровнушкиной*

Подписано в печать 08.11.2022. Формат 70×100 1/16.
Бумага офсетная. Цифровая печать. Усл. печ. л. 7,74.
Уч.-изд. л. 5,1. Тираж 30 экз. Заказ 209.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано Издательско-полиграфическим центром УрФУ
620083, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 358-93-06, 350-58-20, 350-90-13
Факс: 8 (343) 358-93-06
http://print.urfu.ru

Для заметок

Для заметок


УНИВЕРСИТЕТСКОГО КНИГОИЗДАНИЯ
УНИВЕРСИТЕТСКАЯ КНИГА
2020


9 785799 635947