МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«КУЗБАССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Т. Ф. ГОРБАЧЕВА»
Филиал КузГТУ в г. Белово

Кафедра инженерно-экономическая

ПМ.02 Осуществление интеграции программных модулей
МДК.02.01 Технология разработки программного обеспечения

Методические рекомендации
по выполнению практических работ
для специальности
09.02.07 «Информационные системы и программирование»

Составитель: Витвицкий М.Н.
Рассмотрены и утверждены на заседании кафедры
Протокол № 6 от 14.02.2026 г.
Рекомендовано учебно-методической комиссией специальностей СПО в качестве электронного издания для использования в учебном процессе
Протокол № 6 от 17.02.2026 г.

Долганова Жанна Александровна
Подписано цифровой подписью: Долганова Жанна Александровна
Дата: 2026.02.18 11:15:54 +07'00'

Белово 2026

СОДЕРЖАНИЕ

ОРГАНИЗАЦИЯ ПРАКТИЧЕСКОЙ РАБОТЫ

Практическая работа обучающихся может рассматриваться как организационная форма обучения, обеспечивающих управление учебной деятельностью или деятельность обучающихся по освоению общих и профессиональных компетенций, знаний и умений учебной и научной деятельности на примере выполнения тематических практических заданий.

Целью освоения дисциплины «Технология разработки программного обеспечения» является приобретение обучающимися знаний в области формирования требований к информационной системе и тестирования программного обеспечения.

Основными задачами изучения дисциплины «Технология разработки программного обеспечения», являются:

1. Изучение основных принципов процесса разработки программного обеспечения.

2. Изучение встроенных и основных специализированные инструментов анализа качества программных продуктов.

3. использование специализированных графических средств построения и анализа архитектуры программных продуктов.

4. Умение выполнять ручное и автоматизированное тестирование программного модуля.

ПЛАНИРОВАНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ

При разработке рабочей программы по учебной дисциплине или профессиональному модулю при планировании содержания практической работы преподавателей устанавливается содержание и объем учебной информации или практических заданий, которые выносятся на практическую работу, определяются формы и методы контроля результатов. Содержание практической работы определяется в соответствии с рекомендуемыми видами заданий согласно программе учебной дисциплины модуля.

В соответствии с ведущей дидактической целью содержанием практических занятий являются решение разного рода задач, в том числе профессиональных (анализ производственных ситуаций, решение ситуационных производственных задач, выполнение профессиональных функций в деловых играх и т.п.), выполнение вычислений, расчетов, чертежей, работа с измерительными приборами, оборудованием, аппаратурой, работа с нормативными документами, инструктивными материалами, справочниками, составление проектной, плановой и другой технической и специальной документации и др.

Виды заданий для практической работы, их содержание и характер имеют вариативный и дифференцированный характер, учитывают специфику данной дисциплины и индивидуальные особенности студента.

Перед выполнением студентами практической работы преподаватель проводит инструктаж по выполнению задания, который включает цель задания, его содержание, сроки выполнения, ориентировочный объем работы, основные требования к результатам работы, критерии оценки. В процессе инструктажа преподаватель предупреждает обучающихся о возможных типичных ошибках, встречающихся при выполнении задания.

Инструктаж проводится преподавателем за счет объема времени, отведенного на изучение дисциплины.

Практическая работа может осуществляться индивидуально или группами обучающихся в зависимости от цели, объема, конкретной тематики работы, уровня сложности уровня умений обучающихся.

Отчет по практической работе обучающихся предоставляется в электронном виде.

КОНТРОЛЬ РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ ПРАКТИИЧЕСКОЙ РАБОТЫ

Контроль результатов практической работы студентов осуществляться в пределах времени, отведенного на обязательные учебные занятия по дисциплине и практическую работу обучающихся по дисциплине, может проходить в письменной, устной или смешанной форме, с представлением продукта деятельности учащегося.

В качестве форм и методов контроля практической работы обучающихся, могут быть использованы, зачеты, тестирование, самоотчеты, контрольные работы, защита творческих работ и др., которые могут осуществляться на учебном занятии или вне его (например, оценки за реферат).

Критериями оценки результатов практической работы обучающегося являются:

- уровень освоения учащимся учебного материала;
- умение обучающегося использовать теоретические знания при выполнении практических задач;
- сформированность общих и профессиональных компетенций;
- обоснованность и четкость изложения ответа;
- оформление материала в соответствии с требованиями.

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ

ПРАКТИЧЕСКАЯ РАБОТА

Практическая работа - выполнение проверочных упражнений, нацеленные на выявление степени освоенности материала, уровня подготовки студентов, выявление «проблемных сегментов» в рамках дисциплины или темы с целью корректировки дальнейшего плана обучения (дополнительное выделение времени на повторение или тотальное изучение темы и пр.). Данный вид может реализовываться как в групповом порядке (по вариантам), так и в индивидуальном (для каждого студента готовится отдельный пакет заданий: теория и практика).

Общие требования к выполнению работ:

- Все задания выполняются индивидуально
- Обязательное соблюдение сроков
- Документация оформляется согласно стандартам
- Результаты представляются в электронном и печатном виде

Методы оценки работ:

- Текущий контроль
- Промежуточная аттестация
- Итоговая проверка
- Защита практических работ

Рекомендации по оформлению:

- Единый стиль документации
- Структурированность материалов
- Логичность изложения
- Наличие всех необходимых разделов
- Актуальность информации

Система общей оценки практических работ

Критерии итоговой оценки:

- Своевременность выполнения заданий — 10%
- Качество теоретической подготовки — 20%
- Практическая реализация — 30%
- Документационное обеспечение — 20%

- Защита работы — 20%

Шкала оценивания:

- **Оценка «отлично»** — 90-100% выполнения всех критериев
- **Оценка «хорошо»** — 75-89% выполнения критериев
- **Оценка «удовлетворительно»** — 60-74% выполнения критериев
- **Оценка «неудовлетворительно»** — менее 60% выполнения критериев

Дополнительные баллы могут быть начислены за:

- Оригинальность решений
- Практическую значимость
- Качественную презентацию
- Активное участие в обсуждениях
- Своевременную сдачу работ

Темы практических занятий

№ раздела (темы)	Вопросы, выносимые на самостоятельное изучение	Количество часов
ТЕМА 2.1.1. ОСНОВНЫЕ ПОНЯТИЯ И СТАНДАРТИЗАЦИЯ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ.	Практическое занятие №1. «Анализ предметной области»	4
	Практическое занятие №2. «Разработка и оформление технического задания».	4
	Практическое занятие №3. «Построение архитектуры программного средства».	4
	Практическое занятие №4. «Изучение работы в системе контроля версий».	4
ТЕМА 2.1.3. ОЦЕНКА КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ.	Практическое занятие №5. «Уточнение требований: пользовательские истории и критерии приемки»	2
	Практическое занятие №6. «Реализация прототипа и настройка сборки»	2

Практическое занятие 1. Анализ предметной области

Целью работы является изучение методов анализа предметной области. Результатом практической работы является отчет, в котором должны быть

приведены анализ рассматриваемой предметной области и требований к разрабатываемой информационной системе. Для выполнения практической работы № 1 студент должен изучить приведенный ниже теоретический материал. Отчет сдается в электронном (файл Word) виде.

Анализ предметной области

Требования к программному обеспечению (ПО) определяют, какие свойства и характеристики оно должно иметь для удовлетворения потребностей пользователей и других заинтересованных лиц. Однако сформулировать требования к сложной системе не так легко. В большинстве случаев будущие пользователи могут перечислить набор свойств, который они хотели бы видеть, но никто не даст гарантий, что это – исчерпывающий список. Кроме того, часто сама формулировка этих свойств будет непонятна большинству программистов.

Чтобы ПО было действительно полезным, важно, чтобы оно удовлетворяло реальные потребности людей и организаций, которые часто отличаются от непосредственно выражаемых пользователями желаний. Для выявления этих потребностей, а также для выяснения смысла высказанных требований приходится проводить достаточно большую дополнительную работу, которая называется анализом предметной области или бизнес-моделированием, если речь идет о потребностях коммерческой организации. В результате этой деятельности разработчики должны научиться понимать язык, на котором говорят пользователи и заказчики, выявить цели их деятельности, определить набор задач, решаемых ими.

В дополнение стоит выяснить, какие вообще задачи нужно уметь решать для достижения этих целей, выяснить свойства результатов, которые хотелось бы получить, а также определить набор сущностей, с которыми приходится иметь дело при решении этих задач. Кроме того, анализ предметной области позволяет выявить места возможных улучшений и оценить последствия принимаемых решений о реализации тех или иных функций.

После этого можно определять область ответственности будущей программной системы – какие именно из выявленных задач будут ею решаться, при решении каких задач она может оказать существенную помощь и чем именно. Определив эти задачи в рамках общей системы задач и деятельностей пользователей, можно уже более точно сформулировать требования к ПО.

Анализом предметной области занимаются системные аналитики или бизнес-аналитики, которые передают полученные ими знания другим членам проектной команды, сформулировав их на более понятном разработчикам языке. Для передачи этих знаний обычно служит некоторый набор моделей, в виде графических схем и текстовых документов.

Анализ деятельности крупной организации, такой как банк с сетью региональных отделений, нефтеперерабатывающий завод или компания, производящая автомобили, дает огромные объемы информации. Из этой информации надо уметь отбирать существенную, а также уметь находить в ней пробелы – области деятельности, информации по которым недостаточно для четкого представления о решаемых задачах. Значит, всю получаемую информацию надо каким-то образом систематизировать.

Выделение и анализ требований.

После получения общего представления о деятельности и целях организаций, в которых будет работать будущая программная система, и о ее предметной области, можно определить более четко, какие именно задачи система будет решать.

Кроме того, важно понимать, какие из задач стоят наиболее остро и обязательно должны быть поддержаны уже в первой версии, а какие могут быть отложены до следующих версий или вообще вынесены за рамки области ответственности системы. Эта информация выявляется при анализе потребностей возможных пользователей и заказчиков.

Потребности определяются на основе наиболее актуальных проблем и задач, которые пользователи и заказчики видят перед собой. При этом требуется аккуратное выявление значимых проблем, определение того, насколько хорошо они решаются при текущем положении дел, и расстановка приоритетов при рассмотрении недостаточно хорошо решаемых, поскольку чаще всего решить сразу все проблемы невозможно.

Формулировка потребностей может быть разбита на следующие этапы.

1. Выделить несколько основных проблем.

2. Определить причины возникновения проблем, оценить степень их влияния и выделить наиболее существенные из проблем, влекущие появление остальных.

3. Определить ограничения на возможные решения. Формулировка

потребностей не должна накладывать лишних ограничений на возможные решения, удовлетворяющие им. Нужно попытаться сформулировать, что именно является проблемой, а не предлагать сразу возможные решения.

Например, формулировки «система должна использовать СУБД Oracle для хранения данных», «нужно, чтобы при вводе неверных данных раздавался звуковой сигнал» не очень хорошо описывают потребности. исключением в первом случае может быть особая ситуация, например, если СУБД Oracle уже используется для хранения других данных, которые должны быть интегрированы с рассматриваемыми: при этом ее использование становится внешним ограничением. Соответствующие потребности лучше описать так: «нужно организовать надежное и удобное для интеграции с другими системами хранение данных», «необходимо предотвращать попадание некорректных данных в хранилище».

При выявлении потребностей пользователей анализируются модели деятельности пользователей и организаций, в которых они работают, для выявления проблемных мест. Также используются такие приемы, как анкетирование, демонстрация возможных сеансов работы будущей системы, интерактивные опросы, где пользователям предоставляется возможность самим предложить варианты внешнего вида системы и ее работы или поменять предложенные кем-то другим, демонстрация прототипа системы и др.

После выделения основных потребностей нужно решить вопрос о разграничении области ответственности будущей системы, т.е. определить, какие из потребностей надо пытаться удовлетворить в ее рамках, а какие – нет. При этом все заинтересованные лица делятся на пользователей, которые будут непосредственно использовать создаваемую систему для решения своих задач, и вторичных заинтересованных лиц, которые не решают своих задач с ее помощью, но чьи интересы так или иначе затрагиваются ею. Потребности пользователей нужно удовлетворить в первую очередь и на это нужно выделить больше усилий, а интересы вторичных заинтересованных лиц должны быть только адекватно учтены в итоговой системе.

На основе выделенных потребностей пользователей, отнесенных к области ответственности системы, формулируются возможные функции будущей системы, которые представляют собой услуги, предоставляемые системой и

удовлетворяющие потребности одной или нескольких групп пользователей (или других заинтересованных лиц). Идеи для определения таких функций можно брать из имеющегося опыта разработчиков (наиболее часто используемый источник) или из результатов мозговых штурмов и других форм выработки идей.

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

Например, перечень функций может выглядеть так:

1. Все данные о сделках и клиентах будут сохраняться в базе данных.

2. Статус выполнения заказа клиент сможет узнать через Интернет.

3. Система будет поддерживать до 10000 одновременно работающих пользователей.

4. Расписание проведения ремонтных работ будет строиться автоматически.

Предлагая те или иные функции, нужно уметь аккуратно оценивать их влияние на структуру и деятельность организаций, в рамках которых будет использоваться ПО. Это можно сделать, имея полученные при анализе предметной области модели их текущей деятельности.

Имея набор функций, достаточно хорошо поддерживающий решение наиболее существенных задач, с которыми придется работать разрабатываемой системе, можно составлять требования к ней, представляющие собой детализацию работы этих функций.

Соотношение между проблемами, потребностями, функциями и требованиями показано на рис. 1.

Рисунок 1.Соотношение между проблемами, потребностями.

При этом часто нужно учитывать, что ПО является частью программно-аппаратной системы, требования к которой надо преобразовать в требования к программной и аппаратной ее составляющим. В последнее время, в связи со значительным падением цен на мощное аппаратное обеспечение общего назначения, фокус внимания переместился, в основном, на программное обеспечение. Во многих проектах аппаратная платформа определяется из общих соображений, а поддержку большинства нужных функций осуществляет ПО.

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

Задание.

Заданием работы является анализ предметной области и требований к разрабатываемой информационной системе из Приложения 1. Студент может сам предложить задание (предметную область), по аналогии с приведенными примерами.

Контрольные вопросы.

1. Что такое «требование к информационной системе»?
2. Кто занимается выявлением требований к ис?
3. Перечислите этапы формулировки потребностей?

Практическое занятие №2. Разработка и оформление технического задания.

Целью работы является изучение порядка разработки и оформления технического задания (ТЗ). Результатом практической работы является отчет, в котором должны быть приведено разработанное ТЗ.

Для выполнения практической работы № 2 студент должен изучить приведенный ниже теоретический материал. Отчет сдается в распечатанном и электронном (файл Word) видах.

Разработка и оформление технического задания

Техническое задание – это документ, определяющий цели, требования и основные исходные данные, необходимые для разработки автоматизированной системы управления.

При разработке технического задания необходимо решить следующие задачи:

- установить общую цель создания ис, определить состав подсистем и функциональных задач;
- разработать и обосновать требования, предъявляемые к подсистемам;
- разработать и обосновать требования, предъявляемые к информационной базе, математическому и программному обеспечению, комплексу технических средств (включая средства связи и передачи данных);
- установить общие требования к проектируемой системе;
- определить перечень задач создания системы и исполнителей;
- определить этапы создания системы и сроки их выполнения;
- провести предварительный расчет затрат на создание системы и определить уровень экономической эффективности ее внедрения.
- Типовые требования к составу и содержанию технического задания приведены в Таблице 1.

Таблица 1. Состав и содержание технического задания (ГОСТ 34.602-89)

№ п\п	Раздел	Содержание
1	Общие сведения	• полное наименование системы и ее условное обозначение • шифр темы или шифр(номер)договора; • наименование предприятий разработчика и заказчика системы, их реквизиты • перечень документов, на основании которых создается ис • плановые сроки начала и окончания работ • сведения об источниках и порядке финансирования работ • порядок оформления и предъявления заказчику результатов работ по созданию системы, ее частей и

2	Назначение и цели создания (развития) системы	• вид автоматизируемой деятельности • перечень объектов, на которых предполагается использование системы • наименования и требуемые значения технических, технологических, производственно-экономических и др. показателей объекта, которые должны быть достигнуты при внедрении ис
3	Характеристика объектов автоматизации	• краткие сведения об объекте автоматизации • сведения об условиях эксплуатации и характеристиках окружающей среды
4	Требования к системе	Требования к системе в целом: • требования к структуре и функционированию системы (перечень подсистем, уровни иерархии, степень централизации, способы информационного обмена, режимы функционирования, взаимодействие со смежными системами, перспективы развития системы) • требования к персоналу (численность пользователей, квалификация, режим работы, порядок подготовки) • показатели назначения (степень приспособляемости системы к изменениям процессов управления и значений параметров) требования к надежности, безопасности, эргономике, транспортабельности, эксплуатации, техническому обслуживанию и ремонту, защите от внешних воздействий, к патентной чистоте, по стандартизации и унификации • Требования к функциям (по подсистемам): • перечень подлежащих автоматизации задач • временной регламент реализации каждой функции • требования к качеству реализации каждой функции, к форме представления выходной информации, характеристики точности,

		• требования к надежности, безопасности, эргономике, транспортабельности, эксплуатации, техническому обслуживанию и ремонту, защите от внешних воздействий, к патентной чистоте, по стандартизации и унификации • Требования к функциям (по подсистемам): • перечень подлежащих автоматизации задач • временной регламент реализации каждой функции • требования к качеству реализации каждой функции, к форме представления выходной информации, характеристики точности, • перечень и критерии отказов Требования к видам обеспечения: • математическому (состав и область применения мат. моделей и методов, типовых и разрабатываемых алгоритмов) • информационному (состав, структура и организация данных, обмен данными между компонентами системы, информационная совместимость со смежными системами, используемые классификаторы, СУБД, контроль данных и ведение информационных массивов, процедуры придания юридической силы выходным документам) • лингвистическому (языки программирования, языки взаимодействия пользователей с системой, системы кодирования, языки ввода- вывода) • программному (независимость программных средств от платформы, качество программных средств и способы его контроля, использование фондов алгоритмов и программ) • техническому • метрологическому • организационному (структура и функции эксплуатирующих подразделений, защита от ошибочных действий персонала)

№ п\п	Раздел	Содержание
5	Состав и содержание работы по созданию системы	• перечень стадий и этапов работ • сроки исполнения • состав организаций–исполнителей работ • вид и порядок экспертизы технической документации
6	Порядок контроля и	• виды, состав, объем и методы испытаний системы • общие требования к приемке работ по стадиям
7	Требования к составу и содержанию работ по	• преобразование входной информации к машиночитаемому виду • изменения в объекте автоматизации • сроки и порядок комплектования и обучения персонала
8	Требования к документирова	• перечень подлежащих разработке документов • перечень документов на машинных носителях
9	источники разработки	документы и информационные материалы, на основании которых разрабатывается ТЗ и система

Задание.

Заданием работы является разработка технического задания для разработки информационной системы в соответствующей предметной области из Приложения 1.

Контрольные вопросы

1. Что такое техническое задание?
2. Какой ГОСТ регламентирует содержание технического задания?
3. Какие пункты должно содержать техническое задание?

Практическое занятие №3. Построение архитектуры программного средства

Целью работы является изучение порядка проектирования архитектуры ПО. Результатом практической работы является отчет, в котором должны быть приведена разработанная архитектура ПО.

Для выполнения практической работы № 3 студент должен изучить приведенный ниже теоретический материал. Отчет сдается в распечатанном и

электронном (файл Word) видах.

Разработка архитектуры ПО включает рассмотрение следующих этапов:

1. Влияние стандартов на процессы архитектурного проектирования

Осмысленная разработка программных продуктов – комплексное понятие, состоящее из множества высокоинтеллектуальных и взаимосвязанных процессов, таких как:

- анализ требований;
- проектирование;
- кодирование;
- тестирование;

и т.д.

Каждая из обозначенных активностей должна основываться на базе соответствующих стандартов и лучших практик. Дело в том, что подобный тип документов создается на основе наиболее успешного опыта применения специфического вида деятельности, рабочей технологии или артефактов, эталонное описание которых он содержит. Стандарт концентрирует все лучшее и наиболее эффективное, с точки зрения практического достижения конечного результата, подтвержденного массивом статистических данных.

Перед тем как внедрять стандарты в процессы конкретной организации, следует соответствующим образом адаптировать их под реалии конкретной организации.

Должны быть учтены такие аспекты, как:

- ресурсная база организации;
- количество выделенных для деятельности ресурсов, их доступность, квалификация (если речь идет о специалистах), надежность и т. д.;
- сегмент/домен/направление деятельности компании;
- сфера деятельности, с учетом специфических требований к ним со стороны отраслевых/государственных/международных регуляторных органов;
- степень влияния информационных технологий на поддержку и развитие бизнеса;
- инновационность компании и степень участия в инновациях информационных технологий;

и т.д.

Если тема применения и использования стандартов в целях повышения эффективности деятельности Вас заинтересовала, то мы рекомендуем обратиться к моделям зрелости CMMI, методологии SPICE и другим подобным документам.

Перечень стандартов и их содержание на конкретном предприятии должно определяться на стадии планирования.

В момент планирования перечня стандартов, который действительно необходим, целесообразно руководствоваться принципом «золотой середины». Чрезмерное количество внедряемых стандартов требует достаточно большого количества ресурсов. Следует осознавать – внедрение стандарта делает процесс более унифицированным и качественным за счет его регламентации, но при этом, в отдельных аспектах, понижает степень его гибкости. Если компания пришла к пониманию необходимости внедрения определенной практики или стандарта эволюционным путем, то это облегчает и упрощает процесс внедрения и использования соответствующего регламента. Если изменения носят революционный характер, то попытка привнести в организацию избыточное количество стандартов может носить катастрофический характер и остаться «на бумаге», что не является целью их внедрения.

Стандартизация должна обеспечить постепенное повышение качества конкретных процессов (анализ, документирование, кодирование и т. д.), обеспечить прозрачность, однозначность и не противоречивость процессов, минимизацию или устранение появления возможных ошибок при разработке архитектур и программных продуктов.

В качестве отступления можно рассмотреть следующий пример.

Требования к процедуре сертификации авиационных продуктов предусматривают, что в процессе разработки специфического (для авиационных нужд) программного обеспечения, должны использоваться как минимум три стандарта:

Стандарт на работу с требованиями;

Должен описывать методы, правила и инструменты, применяемые для сбора, разработки и управления требованиями, их возможные форматы и нотации.

Стандарт на разработку архитектуры программного продукта;

Содержит правила, формирующие архитектуру программного продукта, приемлемые и неприемлемые методы ее разработки, описывает возможные функциональные и не функциональные ограничения (запрет на использование рекурсии или максимальная вложенность вызовов).

Стандарт процесса кодирования;

Регламентирует исходный код программы, ссылки на описания используемого языка программирования, его синтаксис, ограничение, желательные и нежелательные конструкции языка и прочие правила, касающиеся разработки кода программы;

Стандарты определяют рабочие принципы, которые должны действовать на всем протяжении жизненного цикла программного продукта. Например, если в стандарте зафиксировано, что нельзя вставлять комментарии в программный код (комментарии следует располагать в строго определенном месте) то, все комментарии, расположенные не в соответствии с этим правилом, должны быть перенесены или удалены. Несоответствие принятым стандартам грозит возникновением потенциальных ошибок. Если стандарт по каким-то причинам не соблюдается, то все «нарушения» должны устраняться. Если стандарт требует доработок в связи с изменившимися первоначальными предпосылками его создания, то его необходимо либо дорабатывать, либо менять. Слепое следование стандартам губительно, так же, как и постоянные попытки обойти их по необоснованным причинам.

Проектирование архитектуры программных продуктов – это одна из активностей, для которых должны быть приняты определенные стандарты деятельности. При этом, современные требования к архитектурам предусматривают создание достаточно гибких и адаптивных архитектур.

Первое, «верхнеуровневое» проектирование, должно удовлетворять цели достаточно свободной и абстрактной реализации архитектуры, но при этом учитывать жесткие требования стандартов. Поэтому требования к программным продуктам принято делить на 2 типа критичности – высокоуровневые (функциональные) и низкоуровневые (нефункциональные) требования. Гибкость архитектур реализуется за счет разной степени принципиальности учета требований и их последующей реализации в программном коде будущих информационных

систем. Требования к нефункциональным характеристикам должны быть более четкими и однозначными.

Типы требований мы уже немного обсуждали ранее и будем продолжать это делать походу нашего курса, изучая различные аспекты архитектур и архитектурного проектирования. Сейчас отметим, что подход к работе с ним должен быть также соответствующим образом стандартизован. За счет стандартов важно достичь согласованности между типами требований в момент их согласования друг с другом в виде отдельных компонентов архитектуры или программного продукта. Это позволит добиться непротиворечивого результата, в виде оптимальной архитектуры и эффективного программного обеспечения.

Большинство современных стандартов, описывающих работу с требованиями, выдвигают следующие характеристики, которым должен отвечать каждый из них:

- конкретность/измеримость;
- корректность/обоснованность;
- соответствие функциональности программного продукта;
- возможность последующей верификации;
- уникальность;
- последовательность;
- непротиворечивость;
- возможность декомпозиции;
- изменяемость с минимальными затратами;
- трассируемость.

На сегодняшний день существует множество программных продуктов, которые позволяют автоматизировать и унифицировать процесс разработки и последующего отслеживания требований.

Применение специализированного инструментария, поддерживающего принятые стандарты, в большинстве случаев, положительно сказывается на качестве процессов архитектурного проектирования, разрабатываемых архитектурах и программных продуктов в целом, за счет того, что их использование позволяет выявить те программные компоненты и детали, которые подвержены изменениям меньше остальных. Именно они и составят основу разрабатываемых архитектур, которая будет наименее гибкой, а те требования, которые чаще других изменяются,

будут лежать в основе достаточно гибкой функциональности.Но изменения, иногда вносятся и в «скелет».

Принципиальное следование всем принятым стандартам возможно только тогда, когда речь идет об устоявшихся отлаженных процессах архитектурного проектирования и разработки информационных систем, которые достигли своей наивысшей точки качественного развития.

Принципы и темп развития современного мира определяют его черты. Программные продукты, как часть этих черт должны быть в состоянии поддерживать соответствующую динамику и темпы. Элементы информационных систем и их архитектура предназначены для удовлетворения требований к разрабатываемым продуктам. Стоит понимать, что требования не только могут изменяться, но и качественно трансформироваться, переходя из категории нефункциональных в функциональные и наоборот.

Можно привести следующий пример:

В случае, если речь идет о стадии внедрения ПО, то такая характеристика как сопровождаемость воспринимается пользователями, как некритичная и не значимая с их точки зрения, но как только начинается стадия промышленной эксплуатации программного продукта, затраты на его поддержку и развития сразу дают о себе знать. Хорошо, если компания имеет достаточный для этого ресурс, но если это не так, то поддержка процессов и соответствующего для их выполнения уровня данных «кочует» на плечи бизнес пользователей и сразу переходит в разряд функциональных.

Все вышесказанное явно свидетельствует о значимости и необходимости стандартизированного подхода к созданию новых, инновационных, прорывных информационных продуктов, которые в своей основе будут иметь надежные, качественные, производительные и масштабируемые архитектуры, которые позволят создавать функционально гибкие продукты.

Без четких, ясных требований к выполняемым функциям создать программное обеспечение невозможно. Требования – фундамент каждой разработки.

Процессы разработки должны начинаться с процесса разработки требований и вестись в соответствии с планом, корректируемым по «ходу».

2. Инструментарий разработки и моделирования требований к процессам

и архитектуре программных продуктов

Для перехода от набора требований, порой разрозненных, к стройной и логичной архитектуре программного продукта, нужно использовать специализированный инструментарий. В нашем случае это специальные технологические средства разработки и моделирования процессов, на основе выявленных требований, которые позволят создать единое видение разрабатываемой архитектуры, с учетом определенных правил представления информации, принятой для конкретного проекта по созданию программного продукта или процесса архитектурного проектирования.

Как было отмечено ранее, архитектурное проектирование информационных систем – это область деятельности, которая имеет много общего с архитектурным проектированием в строительстве.

«Строительство» – старший собрат информационных технологий. Поэтому было бы оптимально, для процессов создания программных продуктов многое необходимое для развития данной профессиональной области, почерпнуть, поинтересовавшись тем, как в строительстве успешно решались схожие задачи. В гражданском и промышленном строительстве языком описания архитектуры являются архитектурностроительные чертежи, объемные прототипы и модели, текстовые описания возводимых объектов. Младший собрат поступил правильно и перенял необходимый опыт у старшего. В качестве средств разработки и моделирования характеристик информационных продуктов, в архитектурном проектировании используются различные языки, которые принято называть нотациями, а инструментами, которые поддерживают работу с нотациями – CASE средствами.

Из современных нотаций можно выделить следующие:

Блок-схемы (схемы алгоритмов)

Данный тип схем (графических моделей) визуализирует разрабатываемые алгоритмы или процессы. В описании блок- схем принято отдельные стадии процессов изображать в виде блоков (отсюда и соответствующее название данной нотации). Блоки различаются в зависимости от семантики, содержащейся в представляемой ими форме. Отображаемые блоки соединены между собой линиями, имеющими направление и определенную алгоритмами последовательность.

В широком значении термина «блок-схемы» он является метанотацией. Различные спецификации «блок-схем» получили свое определенное назначение и форму представления необходимой информации, в зависимости от специфики каждой конкретной нотации, но при этом общие правила построения моделей наследуются от правил построения «блок-схем».

DFD-диаграмма (диаграмма потоков данных)

DFD – это нотация структурного анализа.

В данной нотации принято описывать внешние, по отношению к разрабатываемому продукту:

источники данных;

Адресаты данных;

Логические функции;

Потоки данных;

Хранилища данных.

Диаграмма потоков данных – это один из первых и основных инструментов структурного анализа и проектирования верхнеуровневой архитектуры программных продуктов, существовавших до последующего широкого распространения UML.

В основе данной нотации находится методология проектирования и метод построения модели потоков данных проектируемой информационной системы. В соответствии с соответствующей методологией проектирования модель системы идентифицируется как иерархия диаграмм потоков данных, которые представляют собой последовательный процесс преобразования данных, с момента их поступления в систему, до момента представления информации конечному (или псевдоконечному) пользователю.

Диаграммы потоков данных разработаны для определения основных процессов или модулей программного продукта. Далее, в целях комплексного представления разрабатываемой архитектуры, они детализируются диаграммами нижних уровней представления информации (DFD) и процессов (IDEF). Подобный принцип отображения данных продолжается, реализуя многоуровневую иерархию подчиненных и взаимосвязанных диаграмм. В результате будет достигнут нужный уровень декомпозиции, на котором процессы становятся абсолютно понятными и

прозрачными.

В современных процессах разработки программного обеспечения подобный подход к проектированию программных продуктов практически не применяется, по причине смещения акцентов создания информационных систем от структурного к объектно-ориентированному подходу.

Данная методология анализа и проектирования на сегодняшний день эффективно используется в бизнес-анализе и соответствующих дисциплинах, по причине ее наглядности и понятности для стэйкхолдеров.

ER-диаграмма (диаграмма сущность-связь)

ER – это тип нотации, задача которой состоит в создании формальной конструкции, описывающей и визуализирующей верхне-уровневое представление бизнес объектов будущей системы, их атрибуты и связь между ними, в виде основных элементов, которые будут использоваться в качестве фундамента для проектирования таблиц будущей базы данных программного продукта. Таким образом, ER-диаграмма представляется как концептуальная модель создаваемой информационной системы, в терминах конкретной предметной области. Данный тип диаграмм помогает выделить ключевые сущности и определить связи между ними. На сегодняшний день существует достаточно мощный инструментарий, который может позволить выполнить преобразование созданной ER- диаграммы в конкретную схему базы данных на основе выбранной модели данных.

EPC-диаграмма (диаграмма событийной цепочки процессов)

Событийная цепочка процессов – это тип нотации, задача которой заключается в создании моделей бизнес процессов. Она используется для моделирования, анализа и реорганизации бизнес-процессов. Ее отличие, от ранее рассмотренных, состоит в том, что EPC применяют для создания функциональных моделей, имитирующих конкретные процессы, реализуемые в определенном программном продукте.

история развития данного типа диаграмм связана с разработкой одноименного метода в рамках работ по созданию методологии ARIS (Architecture of Integrated Information Systems – Архитектура интегрированных информационных систем).

Диаграмма EPC оформляется в виде упорядоченной последовательности событий и функций. Каждая функция при этом должна определяться начальными и

конечными событиями, могут быть указаны участники, исполнители, материальные и информационные потоки, сопровождающие отдельные функции. Дополнительную ценность диаграммам EPC придает тот факт, что она может использоваться для настройки программных продуктов типа ERP, в части создания или улучшения бизнес- процессов.

BPMN-диаграммы

Один из самых распространенных типов нотаций на сегодняшний день – это BPMN (Business Process Model and Notation).

EPC, ER, BPMN – это не просто нотации, а методы реализации конкретных моделей процессов. BPMN по своему назначению более уместно сравнивать с диаграммами EPC. Причина этого заключается в том, что они имеют схожую цель – функциональное моделирование бизнес процессов. Рассматриваемая диаграмма самый новый, из приведенных в обзоре, принятый стандарт моделирования бизнес процессов. Главное позиционируемое преимущество диаграммы BPMN – понятная всем стэйкхолдерам разрабатываемой архитектуры и программного продукта визуализация моделируемых процессов.

Нотация имеет достаточно ясный инструментарий, позволяющий моделировать как относительно простые, так и сложные бизнес-процессы. Это достигается за счет применения двух групп элементов:

Первая группа (simple notation) состоит из основных элементов BPMN, которые удовлетворяют требованиям создания простой графической нотации. Подавляющее большинство бизнес-процессов моделируются с использованием элементов данной группы.

Вторая группа (powerful notation) содержит полный «пакет» элементов BPMN. В него кроме основных блоков входят специфические комплексные модули, представляющие различные типы и виды элементов, необходимых для моделирования самых сложных процессов.

Подобная преемственность инструментария групп BPMN позволяет удовлетворять требованиям комплексной нотации и управлять более сложными ситуациями моделирования.

Оптимально смоделированные диаграммы BPMN могут применяться для настройки, рефакторинга и реинжиниринга бизнес процессов во многих

современных информационных системах класса ECM.

UML–диаграммы

UML (Unified Modeling Language) – не просто нотация или группа нотаций, а язык графического описания для объектного моделирования.

UML – это язык широкого профиля, который создан для формирования серии нотаций, поддерживающих полный цикл разработки, как архитектуры, так и функциональности программных продуктов, реализуемых по принципам объектно-ориентируемого программирования.

Широкая распространенность и повсеместное применение данного языка привели к тому, что он стал открытым стандартом, использующим графические обозначения для создания моделей систем, которые принято называть UML-моделями. Специфика UML не предполагает ориентацию на описание архитектуры, так как основное его назначение заключается в определении, визуализации, проектировании, документировании, преимущественно программных продуктов, бизнес процессов и структур данных.

Язык UML получил популярность за счет жесткой связки с популярной и распространенной методологией разработки и внедрения программных продуктов RUP, в соответствии с которой организована работа многих консалтинговых и производственных компаний отрасли информационных технологий. Не смотря на множество преимуществ, UML не является панацеей и абсолютным универсумом в процессах проектирования архитектур и программных продуктов.

Применять и разбираться в диаграммах UML могут только узкоспециализированные специалисты области проектирования программных систем. Вовлечь в обсуждение и согласование разрабатываемой архитектуры информационного продукта, задокументированного на UML, стэйкхолдеров, представителей не направления информационных технологий, будет достаточно сложно.

UML, в отличие от ER и BPMN диаграмм, поддерживает генерацию не просто отдельной функциональности, привязанной к специфичным типам информационных систем, а целостного программного кода на основании сформированных UML-моделей.

EIP-диаграммы

Пожалуй самая специфичная и наименее распространенная нотация.

EIP (EnterpriseIntegrationPatterns) диаграммы –это набор из 65 шаблонов диаграмм, которые предназначены для описания специализированных процессов взаимодействия между программными продуктами. Специфика данной нотации состоит в том, что она разработана для описания конкретных процессов, обеспечивающих интеграционное взаимодействие между приложениями и функциональность программных продуктов, ориентированных на обмен сообщениями между информационными системами.

Шаблоны интеграции приложений и необходимая функциональность, описание которых поддерживают EIP диаграммы, встроены в множество современных специализированных открытых программных продуктов и доступны для применения в специализированных интеграционных процессах.

На сегодняшний день, в силу своей специфичности, данный вид нотации не имеет широкого распространения, но, учитывая темпы роста популярности интеграционных процессов, можно спрогнозировать, что в ближайшее время интерес к EIP- диаграммам вырастет.

Модели компонентов, программных продуктов и функциональных процессов

Модели, как таковые, не являются нотациями, а представляют собой набор необходимых для реализаций процессов/подпроцессов, представленных в виде нескольких взаимодополняющих друг друга нотаций, с целью моделирования конечного результата. Говоря о модели, со структурной точки зрения, корректно будет сказать, что это «метанотация», включающая в себя несколько основных нотаций, реализация диаграмм которых запланирована для качественной разработки конкретной архитектуры.

Таким образом, модель – это абстракция, которая фокусирует субъект, работающий с ней на основных, критичных для реализации, характеристиках конечного продукта. Из примеров CASE средств, способных реализовать и представить нотации в виде модели можно выделить ARIS, UML.

Так же возможна схема, когда исполнитель разработает необходимые диаграммы в разных CASE средствах, после чего компилирует модель самостоятельно. В таких случаях дальнейшее сопровождение и развитие модели

затруднительно (сложность сопровождения, развития, управления моделью) и целиком и полностью лежит на «плечах» ответственных за это специалистов.

Прототипы

Когда мы говорили о моделях, то отметили, что модель – это следующий, качественный шаг вперед к реализации программного обеспечения, по сравнению с диаграммами, реализуемыми нотациями.

После того, как реализованы несколько моделей требований, можно приступать к реализации прототипов, если это предусмотрено и спланировано в проекте разработки программного продукта.

Прототип представляет собой «черновой» вариант конечной реализации программного продукта или его компонента, разработанного с учетом определенных ограничений для демонстрации за- интересованным сторонам на определенной стадии разработки только архитектуры или программного продукта в целом.

Цель разработки прототипа – подтвердить ожидания стэйкхолдеров от реализации конечного продукта и согласовать ее функциональность. Прототипы принято реализовывать в специализированном программном коде, поэтому их следует воспринимать как часть процесса разработки, но при этом не рассчитывать на то, что прототип будет являться частью будущей архитектуры или информационной системы.

Прототипы, как правило, разрабатываются достаточно быстро с помощью специализированного инструментария, освоение которого требует гораздо меньше времени и сил, чем освоение определенного языка программирования.

Жизненный цикл прототипа заканчивается в тот момент, когда ожидания стэйкхолдеров подтверждены и разработанный облик (не более) будущего продукта запланирован к реализации.

Интерфейсы:

Работа по созданию интерфейсов – это отдельная область разработки программных продуктов, которая не определяет архитектуры, но в большинстве случаев зависит от нее.

Эта область требует к себе довольно много внимания и сил. Она определяет многие функциональные характеристики программных продуктов (эргономика, дизайн интерфейсов, воспринимаемость и удобство работы с информацией и т. д.).

Значимость корректных и эффективных интерфейсов все больше и больше растет для разнообразных программных продуктов. Интерфейс представляет собой «верхушку», с помощью которой пользователь будет взаимодействовать с «нутром» программного продукта.

Важно, чтобы уже на ранних стадиях разработки интерфейсов пользователь имел представление о том, с чем и каким образом ему придется работать и постепенно привыкал, при возможности корректируя его или свои ожидания от него.

На текущий момент есть множество инструментов для создания интерфейсов разной степени сложности и детальности. Мы выделим следующие Balsamiq mockup, Axure RP и пр.

Каждый крупный программный продукт или его модуль, который состоит из совокупности компонентов и связей между ними, должен быть детально описан в соответствующий момент процесса архитектурного проектирования. Описание должно быть выполнено по стандартизированным правилам организации или проекта с использованием необходимых нотаций для данного конкретного случая.

По принципам программной инженерии и здравой логики, которая находится в основе процессов разработки информационных систем, любая подсистема или компонент в дальнейшем может выступать в качестве составного элемента более масштабной системы. Поэтому в архитектурном проектировании должно обязательно содержаться подробное описание укрупненных частей системы, выполненных с помощью CASE-средств и нотаций, согласованных друг с другом и обеспечивающих последующую преемственность.

Важно, чтобы выбранная нотация или серия нотаций поддерживали существующий процесс архитектурного проектирования, способствовали его развитию и совершенствованию и были «в силах» поддержать соответствующую фиксацию функциональных, нефункциональных и прочих требований к разрабатываемой архитектуре и программному продукту в целом.

Необходимо осознавать, что нет единой нотации или языка проектирования, который мог бы быть решением всех

поставленных задач процесса проектирования. Сегодня существует достаточное количество универсальных инструментов и средств, но ни один из них

не может обеспечить весь спектр возникающих задач проектирования.

Для того, чтобы создать достаточно комплексную и взаимосвязанную среду проектирования архитектур, моделей и функциональности программных продуктов нужно использовать инструменты и диаграммы, которые смогут поддержать процессы разработки, в зависимости от специфики конкретной задачи, но при этом должен существовать минимальный набор необходимых диаграмм, который должен присутствовать в каждом проекте по разработке программного продукта не смотря на его масштабы и значимость.

3. О функциональных и нефункциональных требованиях к архитектуре и функциональности программного обеспечения

Рассмотрим подробно функциональные и нефункциональные требования к архитектуре программного обеспечения.

Принято думать, что архитектура формируется под воздействием, прежде всего, нефункциональных требований, которые, в большинстве случаев, отсутствуют в проектируемых функциональных и бизнес моделях систем, но начинают свой жизненный цикл в процессе реализации программных продуктов. Попробуем обосновать, что это не совсем верно.

Многие современные ученые области инженерии требований высказывают мнение о том, что не бывает нефункциональных требований.

Нефункциональные требования – это абсолютно функциональные требования, которые описывают функции системы с точки зрения «каких-то» стэйкхолдеров, о которых забыли в процессе сбора и анализа требований. Вы знаете какое- нибудь программное обеспечение, в процессе разработки которого активно участвовали будущие специалисты групп сопровождения, тестирования, развития его архитектуры и функциональности? Как часто проектируя программные продукты о многих его характеристиках, не значимых на первый взгляд для бизнес-стейкхолдеров, но критичных для других

пользователей, просто забывают или намеренно игнорируют, считая их не значимыми и второстепенными.

Ранее мы показали, что члены группы поддержки, тестирования, системные администраторы и некоторые другие технические специалисты являются такими же стэйкхолдерами, как и будущие основные бизнес пользователи системы.

Игнорирование их требований может привести к тому, что фаза внедрения пройдет достаточно успешно, но вот последующее сопровождение и развитие системы может быть достаточно проблематичным.

Основная задача заключается в том, что для современного бизнес мира «последующее» значит то, о чем можно забыть сегодня и не вспоминать до тех пор, пока забытое не напомнит о себе. В случае с нефункциональными требованиями такой подход к работе может оказаться слишком дорогостоящим. Как только информационный продукт перейдет в стадию промышленной эксплуатации и над ним начнут работать группы специалистов, которые должны будут поддерживать достаточно высокий уровень сервисной поддержки и развития системы, может оказаться, что его реализация не включает в себя множество разнообразных технических аспектов. Ведь именно от таких аспектов зависит качество продукта, обеспечивающее оптимальность бизнес-процессов, моделей и данных. Довольно распространенная ситуация, когда на ранних фазах создания программных продуктов о таких характеристиках как надежность, быстродействие, безопасность многие специалисты и стэйкхолдеры, вовлеченные в процесс проектирования архитектуры и функциональности просто не задумываются, отдавая их на дальнейшую проработку и реализацию разработчикам. Все бы хорошо, но для разработчиков существует множество факторов, в соответствии с которыми, модели программных компонентов, которые не имеют четких требований будут реализованы не так как задумывалось при составлении специализированных документов, а так, как разработчик «сможет». Это «сможет» будет определяться квалификацией, инструментарием и, конечно же тем количеством времени, которое у него/них будет в наличии, а его, как известно, практически всегда не хватает. Таким образом, получается следующий парадокс – качество продукта будет напрямую зависеть не от стэйкхолдеров, а от специалистов, для которых важность продукта, который они разрабатывают, не очевидна. Стэйкхолдерам важно получить законченную функциональность в сроки согласованных работ, а иногда даже намного, намного ранее и получить дополнительное время на реализацию качественных атрибутов, значимость которых определяется только в процессе разработки. Именно поэтому многие характеристики разрабатываемого программного продукта и его архитектуры, скрытые от внимательного взора

руководства, необходимо обозначать в активностях, предваряющих стадии проектирования и разработки. Многое на этой стадии зависит от опыта и мастерства системного архитектора и его команды, от профессионализма которых будет зависеть стратегический успех информационной системы, разрабатываемой для бизнес необходимостей конкретной организации.

Из сказанного выше следует, что требования не бывают второстепенными или незначимыми. Незначимое сегодня окажется жизненно необходимым завтра. Рамки работ над требованиями должны быть спланированы и ясно понятны всем стэйкхолдерам, участвующим в проекте.

После того, как мы определились с тем, что для создания качественной архитектуры не должно быть разного отношения к функциональным и не функциональным требованиям. Все требования необходимо рассматривать с точки зрения их критичности по отношению к архитектуре программного продукта.

Высокоуровневое назначение программного продукта – приносить выгоду его владельцу за счет автоматизации труда (интеллектуального, физического, монотонного и т.д.) человека, а в идеале за счет полной замены человеческого ресурса и фактора, если это возможно.

Здесь будет уместно сравнение со строением человека, в котором есть множество органов, каждый из которых отвечает за определенную функцию и каждый развивается не только как самостоятельная единица, но и как часть целого. В тот момент, когда определенный орган дает сбой и не может поддерживать свою целевую функциональность, при условии что это не отражается на деятельности всего организма, то можно считать, что это не критично, но если весь организм выходит из строя на определенный период, то важно понять в чем состоит источник проблемы и устранить его, а если подойти системно, то не допустить подобной возможности. Таким образом, те элементы органов, которые влияют на их полную функциональность и взаимосвязаны с другими, являются наиболее приоритетными для исследования и оптимального содержания и развития. Подобными компонентами в программном продукте являются компоненты, наиболее критичные для рассмотрения с точки зрения архитектуры и сбора требования.

Но при разработке определенной информационной системы, классификация требований необходима для целей дальнейшей их трассировки и управления ими.

Традиционно выделяют 2 основные группы описание которых и более подробную литературу по работе с которыми можно найти в соответствующей литературе.

Сначала рассмотрим функциональные требования.

Функциональные требования описывают «поведение» системы и информацию, с которой система будет работать. Они описывают возможности системы в терминах поведения или выполняемых операций

Цель использования данной группы требований (как следует из названия) заключается в регламентации возможностей и соответствующем поведении разрабатываемого программного продукта. Функциональные требования должны отвечать на вопрос – каким образом должны быть алгоритмизированы процессы информационного продукта, чтобы взаимодействие между пользователем и системой удовлетворяло потребности стэйкхолдеров?

Именно с помощью функциональных требований, в большинстве случаев, определяются рамки работ по процессам, сопровождающим цикл создания программных продуктов. Данный тип требований устанавливает:

Цели разрабатываемого функционала;

Задачи, которые должны быть выполнены для достижения поставленной цели;

Сервисы, поддерживающие выполнение задач;

… (и многое другое, касающееся непосредственно функциональных возможностей и особенностей программного продукта).

На сегодняшний день существует множество подходов к разработке и фиксации функциональных требований. Кратко рассмотрим наиболее популярные из них:

Классический подход

Суть классического подхода состоит в разработке требований с помощью итеративной работы с верхнеуровневыми требованиями стэйкхолдеров, и постепенной детализации до уровня понятного разработчикам. Это наиболее изученный и популярный подход, который подробно описан в современной литературе.

Дополнительно следует отметить, что в подобном подходе основная тяжесть создания полноценного документа, охватывающего весь программный продукт, ложится на сотрудника, ответственного за сбор, анализ и синтез информации. При

современной динамике изменений бизнес процессов и данных, поступающих от внешних и внутренних аспектов бизнеса, непосредственно влияющих на требования стэйкхолдеров, классический подход становится наименее эффективным. Именно поэтому в последнее время водопадные (каскадные) модели разработки программного обеспечения заменяются «гибкими» подходами (agile), цель которых в быстром и эффективном решении возникающих задач, даже если следующая будет противоречить предыдущей.

Usecases (Варианты использования);

В этом подходе функциональные требования записываются с помощью системы специализированных правил, которые, к примеру, должны фиксироваться следующим образом: «программный продукт должен обеспечить учетчику возможность формирования акта выдачи бланков строгой отчетности».

Если определить максимальное количество возможных вариантов использования разрабатываемого программного продукта предполагаемыми целевыми пользователями, то мы получим достаточно полные функциональные требования.

Но, при попытке тотального применения к работе над функциональными требованиями этого подхода существует вероятность того, что отдельные, незначимые для определенных стэйкхолдеров, потребности будут не учтены. В этом случае существует вероятность упустить суть конкретных функциональных требований, которые, как правило, скрыты от постороннего взгляда. Чтобы этого не произошло, важно использовать в обработке зафиксированных use-cases классический подход, в рамках которого должен быть проведен анализ, систематизация и синтез информации. Это поможет выявить истинное назначение предполагаемого к реализации функционала и не разрешить отдельных деталей.

Нефункциональные требования, в дополнение к функциональным, направленны на обеспечение технической целостности разрабатываемого функционала и поддержку характеристик реализуемого программного обеспечения, которые необходимы для создания оптимальной архитектуры.

Они регламентируют внутренние и внешние условия функционирования программного продукта. Выделяют следующие основные группы нефункциональных требований:

- Атрибуты качества;
- Безопасность;
- Надежность;
- Производительность;
- Скорость и время отклика приложения;
- Пропускная способность workflow;
- Количество необходимой оперативной памяти;
- Ограничения;

Платформа реализации архитектуры и программного продукта;

Тип используемого сервера приложений.

Нефункциональные требования описывают условия, которые не относятся к поведению и функциональности системы, но обеспечивают их на уровне компонентов архитектуры программного продукта.

Ранее описанная методология use-cases может быть применена для сбора и анализа нефункциональных требований. При взаимодействии со стэйкхолдерами необходимо фиксацию каждого правила подытоживать фиксацией нефункциональных требований, выраженную в виде количественной характеристики определенного параметра: «программный продукт должен обеспечить учетчику возможность формирования акта выдачи бланков строгой отчетности за время не более 0,5 секунды после того, как поступила соответствующая команда»

4. Прочие виды требований

Любое предприятие, которое осознанно пришло к необходимости создания или применения программного продукта, обладающего оптимальной архитектурой, для определенных условий и аспектов деятельности, можно считать сложной организационно-технической системой, которая имеет определенные цели и реализует конкретные функциональные задачи. При предпосылках использования программных продуктов в условиях организационного управления, процесс исполнения каждой задачи нуждается в конкретных ресурсах для реализации цели функционирования предприятия.

Сложная организационно-программная система состоит из иерархических уровней и поддерживающих их программных структур (исполнители бизнес

процессов и необходимые программные продукты), каждый из которых постоянно расширяется и развивается. Из этого следует, что развитие каждого отдельного элемента (новая версия информационной системы с исправленными ошибками и дополнительными функциональными возможностями или развитие конкретного исполнителя) будет приводить к улучшению характеристик программного продукта, используемого на конкретном предприятии. Но целенаправленное развитие можно будет обеспечить только при условии соблюдения требования взаимодействия всех составных компонентов организации.

Запланированное функционирование бизнес процессов, поддерживаемых архитектурой программных продуктов, необходимо обеспечить требованиями к ресурсам, которые могут быть кратко выражены следующими аспектами:

Время:

Время на обучение пользователей, стабилизацию реализованных и внедренных принципов работы, поддерживающих достижение целей бизнес процессов, время на осознание организацией «хозяйнических» инстинктов по отношению к программному продукту.

Финансы:

Финансовые ресурсы должны обеспечить необходимый уровень поддержки, выраженные в оптимальном количестве квалифицированных исполнителей их профессиональной мотивации, требуемом уровне технической оснащенности и пр.

Данные и информация:

Для того, чтобы результат деятельности соответствовал ожиданиям стейкхолдеров требуются данные надлежащего качества, на основе которых стало бы возможным достижение поставленных целей и качественной трансформации данных в значимую для бизнеса информацию. В дальнейшем повторное использование подобной информации позволит повысить уровень ценности тактических и стратегических процессов и способствовать повышению уровня и зрелости компании в целом.

Если перейти от организационных требований к реализации архитектуры программных продуктов, общая специфика большинства современных практик процессов создания архитектур, то можно выявить ряд общих особенностей, которые заключаются в:

1. Преимущественной ориентированности на водопадные модели процессов реализации архитектур и программных продуктов;

2. Подавляющем фокусировании на архитектуре программного продукта и практически полном игнорировании того факта, что система включает в себя не только информационно-программные компоненты, но и другие аспекты (технические средства, персонал), которые также должны быть рассмотрены и учтены при проектировании решения;

3. Отделение активности технико-экономического обоснования реализации программного продукта от разработки бизнес-процессов и разработки архитектуры системы.

После детального и компетентного изучения специализированных методик результатом их осмысления можно сформулировать следующие требования к процессам проектирования и реализации архитектуры, в частности, и программных продуктов в целом:

Проектирование и создание архитектуры, бизнес-анализ, технико-экономическое обоснование создания продукта, моделирование процессов должны быть неразрывно связаны друг с другом и изменение одного из составляющих должно запускать процессы анализа влияния и управления изменениями.

Сущность процессов проектирования и разработки архитектуры, функциональности программных продуктов должна быть преимущественно итерационной.

В практике создания программных продуктов, каждая реализуемая информационная система имеет собственную специфику, выраженную определенными условиями и факторами, которые имеют различную природу возникновения и силу влияния на архитектуру и функциональность.

Активность инженерии требований, которая ставит своей целью работу с требованиями – это отдельная область, которая нуждается в подробном рассмотрении. Если Вас заинтересовало данное направление отрасли информационных технологий, вы сможете самостоятельно приступить к ее изучению, используя общедоступные источники информации.

5. Зависимости и связи между различными видами требований, функциональности и архитектуры программного обеспечения

После того, как мы определились с основными видами требований, которые необходимо фиксировать, формализовывать и реализовывать в процессах проектирования и разработки для целей реализации архитектуры и функциональности программных продуктов, стоит задуматься о соблюдении целостности архитектуры, необходимой для:

1. Создания оптимального функционала;

2. Реализации значимых аспектов разрабатываемого программного обеспечения;

3. Разработки кросс бизнес-процессов, поддерживаемых архитектурой и функциональностью программного продукта;

4. Взаимосвязи между различными стадиями процессов разработки и внедрения программных продуктов (интеграция, миграция данных и пр.).

Каждое требование, зафиксированное в целях создания информационной системы, походу стадий проектирования, разработки, тестирования и внедрения программного продукта должно быть трансформировано в определенный программный модуль, тестовую процедуру, пункт инструкции пользователя и т. д. – это один из основных постулатов создания качественного и адекватного программного продукта, который называется трассирование (трассировка).

Трассирование представляет собой процесс или атрибут в рамках реализации информационный системы, который обеспечивает связь между его элементами и функциональными процессами. Трассировка должна способствовать установлению связи между:

всеми видами требований;

функциональными и нефункциональными процессами;

результатами;

необходимой отчетностью.

Оптимально выстроенный процесс трассирования должен ясно и однозначно позволять понять что(?), откуда(?), каким образом(?) вышло, и куда (?), и в каком виде (?) поступило.

К примеру, можно привести стандарты создания программных продуктов для авиационной промышленности. В них зафиксировано, что команда, задействованная в разработке программного продукта должна в любой момент времени жизненного

цикла программного продукта уметь отследить цепочку от результатов тестирования, к самим тестам, от тестов к бинарному коду, от бинарного кода к исходному коду, от исходного кода к низкоуровневым требованиям, от низкоуровневых требований к архитектуре, от архитектуры к высокоуровневым требованиям, от высокоуровневых требований к системным требованиям и в обратную сторону. Но, как правило, в стандартах не определяется способ трассирования. Это оставляет системным архитекторам или другим специалистам, ответственным за создание информационных систем, определенную свободу действий, регламентированную принципами здравого смысла и необходимостью получения оптимального результата для конкретных заданных условий.

Когда мы говорим о трассируемости документов, содержащих требования к разрабатываемому продукту, то вполне достаточно обеспечить начальную связность на первых этапах разработки за счет ссылочности создаваемых документов и уникальной идентификации каждого конкретного требования. Если же мы начнем обсуждать непосредственно процессы проектирования, разработки требований и кода программного обеспечения, то необходимо осветить процессы валидации, верификации и тестирования.

Под валидацией понимается процесс, направленный на доказательство того, что верхнеуровневые требования стэйкхолдеров будут полностью удовлетворены и покрыты в разработанной функциональности программного обеспечения.

Верификация является активностью процесса валидации, цель которой проверка и последующее достижение соответствия между требованием и реализованными архитектурой и функциональность программного обеспечения.

Тестирование представляет собой «сугубо техническую» часть активности верификации, направленную на испытание программного продукта.

При выполнении тестирования должны быть решены 2 основные задачи:

1. Демонстрация соответствия требований реализации программного продукта;

2. Выявление ситуаций и аспектов, в которых функциональность и архитектура является несоответствующим зафиксированных в документах требованиям с последующим выполнением п. 1.

Необходимо четко представлять, что ни один из представленных процессов,

сам по себе, не обеспечит качественной трассируемости требований с момента их фиксации до момента промышленной эксплуатации программного продукта. Даже наиболее развитый и формализованный процесс тестирования программного обеспечения не позволяет однозначно, точно и полностью выявить все несоответствия и установить адекватную функциональность, необходимую для достижения поставленных результатов. Причины этого заключаются в множестве разнообразных факторов, основной из которых это пресловутый человеческий фактор, к более одробному рассмотрению которого мы периодически возвращаемся в процессе изучения нашего курса.

Только в совокупности, используя свойства, порожденные эффектом эмерджентности, можно добиться достижения системных результатов от процессов трассировки.

При выполнении каждой стадии работы над требованиями и разрабатываемой архитектуры и функциональности информационной системы, в процессы проектирования и реализации должны быть вовлечены все стэйкхолдеры, которые при необходимости смогут прояснить проблемную ситуацию, в рамках зафиксированных требований. Относительно частая связь между разработчиками и стэйкхолдерами будет способствовать:

- повышению прозрачности процессов проектирования и разработки для влиятельных стэйкхолдеров;
- эффективности процессов взаимодействия и оптимизации связей между различными членами команды, заинтересованными сторонами и другими вовлеченными в общие процессы взаимодействия пользователей.

Эти факторы в совокупности приведут к повышению степени доверия между исполнителями и заказчиками и, как следствие, будут позитивно влиять на степень удовлетворенности пользователей и эффективность разрабатываемого программного продукта.

Тема трассирования, как и многие рассматриваемые в нашем курсе, является полноценной областью сферы информационных технологий, для успешного изучения которых требуется затратить определенное время, при этом уже обладая определенным багажом знаний и опыта, полученного во время практической работы над созданием информационных продуктов.

- **. Риски реализации архитектуры и методы управления ими**

Риск – это потенциальная возможность наступления вероятного события/явления или их совокупности, которые могут вызывать определенное влияние (негативное или позитивное) на осуществляемую деятельность или реализуемый продукт.

Под рисками реализации архитектуры понимается совокупность факторов, управленческих решений и других аспектов, которые могут оказать влияние на конечный результат разработки архитектуры и функциональности программных продуктов.

Как правило, риски реализации архитектуры можно ассоциировать со следующими причинами их возникновения:

Попытка создания оптимальной архитектуры на основе уже существующей, но не удовлетворяющей ожидания заказчиков;

Довольно популярная ситуация, складывается в процессе вынужденного реинжиниринга, когда без обоснованного анализа причинно-следственных связей, разработчик или архитектор спонтанно принимают решение о переделке какого-то компонента. В данном случае риск заключается в том, что идея не перешла на стадию «выдержанного» предложения, а сразу начала претворяться в код. Через какое-то время становится понятно, что какой-то элемент остался недостаточно проработан. В таком случае, достаточно «благостном», приходится отказываться от реализации, но если какие-то части уже внедрены в функционирование и ждут своих доработок, то приходится в режиме пожарника искать «обходные» варианты и усугублять имеющуюся информационную систему в целом.

В процессе реализации архитектуры программного продукта меняются ключевые участники команды, задействованной в работе над ней;

В том случае, когда новые члены команды, приступающие к работе над архитектурой и функциональностью программного продукта первым делом, не изучив причин реализации и их следствия, начинают недоумевать по поводу выбранного способа реализации. Им кажется, что все сделано не оптимально, не очевидно, слишком сложно. После того, как выполнены определенные доработки, в процессе тестирования начинают появляться дыры в уже разработанном решении и архитектура из целостного монолита становится «костыльной». В итоге систему

требуется переделать почти полностью. Чем старше система, тем таких ситуаций больше.

Описанные ситуации, как и множество других, особенно выделяются в длительных проектах.

На текущий день, в области создания информационных систем накоплен достаточно большой опыт, на основе которого возможно снижение, локализация или устранение вероятности наступления подобных ситуаций. Среди рисков реализации архитектуры следует выделить следующие, на которых мы сконцентрируемся в дальнейшем:

- риск смены разработчика;
- риск ошибочно принятого архитектурного решения.

Эти риски сильно влияют на предсказуемость процессов перепроектирования и разработки программного продукта. Что характерно, наступление первого усиливает вероятность наступления второго.

Риск смены разработчика связан с потерей ключевой информации об архитектуре и функциональности по причинам изменения ключевого лица. Многие мысли, воплощенные в коде, которые не нашли отражение на бумаге, не всегда понятны тому, кто не является их автором. Подобных рисков можно избежать полным документированием выбранных решений до того момента, как они будут применены. Это позволит относительно безболезненно единовременно выдержать смену любых участников команды. Ключевым фактором успеха становится поддержка системности процессов документирования.

Причины рисков ошибочно принятых архитектурных решений заключаются в недостаточной трассируемости требований от стадии сбора информации, к стадии проектирования или же в недостаточной компетенции лиц, принимающих важные технические решения.

Для того, чтобы максимально обезопасить программный продукт от подобного рода рисков, необходимо как можно более комплексно подойти к решению некоторых вопросов еще на уровне формирования идеи. Все важные решения, связанные с проектированием, разработкой и последующим изменением архитектуры и функциональности программного продукта необходимо мысленно приостанавливать и стремиться задокументировать в виде концептуально

проработанного предложения. Подобный подход помогает не только в принятии обоснованного и согласованного, с уже созданными компонентами информационно системы, решения, но и является своего рода дополнительной тренировкой аналитических способностей.

После того как идея получает свою материализацию на бумаге, в большинстве случаев, она может немного трансформироваться и становится более обдуманной и обоснованной.

Этот метод позволяет подойти к необходимым доработкам с позиции необходимых и достаточных трудозатрат и не тратить ценные ресурсы на «холостые ходы».

Каждое зафиксированное изменение в последующей стадии анализа и проектирования должно быть соотнесено с уже существующей архитектурой с помощью активности анализа влияния. Для этого есть множество разнообразных инструментов от общения с ответственными разработчиками, до построения диаграммы причинно следственных связей, на которых должно быть продемонстрировано то, как предлагаемое изменение будет влиять на программный продукт в целом. Если конструктивных моментов больше и изменение будет влиять положительно, то незамедлительно стоит переходить к его реализации, в противном случае имеет смысл отложить ее или поискать более выигрышные варианты изменений.

В процессе работы над рисками важно помнить первоначальные цели, достижение которых является главным результатом нашей деятельности. Иногда бывает, что увлекшись поиском обходных решений и пытаясь минимизировать величину возможного ущерба, проектная команда, занятая поиском наиболее выигрышных ответов, отклоняется в сторону, и архитектура становится расплывчатой и слишком гибкой, что впоследствии отражается на ее характеристиках и функциональности программного продукта. Кроме того, необходимо учитывать имеющиеся ресурсы, технологии и возможности бизнеса. Программный продукт в процессе своего функционирования должен поддерживаться и обеспечиваться в соответствии с тем количеством ресурсов, которое предусматривается для конкретной стадии его использования.

В том случае, если походу разработки программного продукта

первоначальные предпосылки его создания изменились, подобная ситуация является возможностью для появления или увеличения количества и степени проявления всевозможных рисков (не только архитектурных, но и бизнес, операционных и т. д.).

Поэтому важно контролировать:

1. Детали реализации архитектуры и функциональности информационной системы;

2. Влияние принятых изменений:

Не только на сам продукт, но и на величину ресурсов, необходимых для разработки и сопровождения (как бизнес, так и технических) принятых изменений;

3. Ограничения, которые сопровождают:

Разработку программного продукта;

Жизненный цикл программного продукта с момента его выхода на рынок.

Задание

Заданием работы является проектирование архитектуры программного обеспечения для разработки информационной системы в соответствующей предметной области из Приложения 1.

Контрольные вопросы

1. Какие технологии для проектирования архитектуры ПО существуют?
2. По каким принципам начинают разрабатывать архитектуру ПО?

Практическое занятие №4. Изучение работы в системе контроля версий.

Целью работы является изучение порядка работы с системой контроля версий GIT. Результатом практической работы является отчет, в котором должны быть приведено описание созданного репозитория, демонстрация приемов работы с ним.

Для выполнения практической работы № 4 студент должен изучить приведенный ниже теоретический материал. Отчет сдается в распечатанном и электронном (файл Word) видах.

Изучение работы в системе контроля версий

Git – это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах (чаще всего речь идет об исходном коде программ, но вы можете использовать его для любых файлов на ваш вкус). С его помощью вы можете откатиться на более старую версию вашего проекта, сравнивать, анализировать, сливать изменения и многое другое. Этот процесс называется

контролем версий. Существуют различные системы для контроля версий. Вы, возможно, о них слышали: SVN, Mercurial, Perforce, CVS, Bitkeeper и другие.

Git является распределенным, то есть не зависит от одного центрального сервера, на котором хранятся файлы. Вместо этого он работает полностью локально, сохраняя данные в папках на жестком диске, которые называются репозиторием. Тем не менее, вы можете хранить копию репозитория онлайн, это сильно облегчает работу над одним проектом для нескольких людей. Для этого используются сайты вроде github и bitbucket.

Установка

Установить git на свою машину очень просто:

Windows – мы рекомендуем git for windows, так как он содержит и клиент с графическим интерфейсом, и эмулятор bash.

Настройка

Итак, мы установили git, теперь нужно добавить немного настроек. Есть довольно много опций, с которыми можно играть, но мы настроим самые важные: наше имя пользователя и адрес электронной почты. Откройте терминал и запустите команды:

gitconfig--globaluser.name«MyName»

gitconfig--globaluser.emailmyEmail@example.com

Теперь каждое наше действие будет отмечено именем и почтой. Таким образом, пользователи всегда будут в курсе, кто отвечает за какие изменения – это вносит порядок.

Создание нового репозитория

Как мы отметили ранее, git хранит свои файлы и историю прямо в папке проекта. Чтобы создать новый репозиторий необходимо открыть терминал, зайти в папку своего проекта и выполнить команду init. Это включит приложение в этой конкретной папке и создаст скрытую директорию .git, где будет храниться история репозитория и настройки. Создайте на рабочем столе папку под названием git_exercise. Для этого в окне терминала введите:

$mkdir Desktop/git_exercise/

$cdDesktop/git_exercise/

$gitinit

Командная строка должна вернуть что-то вроде:

InitializedemptyGitrepositoryin/home/user/Desktop/git_exercise/.git /

Это значит, что наш репозиторий был успешно создан, но пока что пуст. Теперь создайте текстовый файл под названием hello.txt и сохраните его в директории git_exercise.

Определение состояния

status – это еще одна важнейшая команда, которая показывает информацию о текущем состоянии репозитория: актуальна ли информация на нем, нет ли чего-то нового, что поменялось, и так далее. Запуск git status на созданном репозитории должен выдать:

$git status

Onbranchmaster Initial commit Untracked files:

(use«gitadd...»toincludeinwhatwillbecommitted) hello.txt

Сообщение говорит о том, что файл hello.txt не отслеживаемый. Это значит, что файл новый и система еще не знает, нужно ли следить за изменениями в файле или его можно просто игнорировать. Для того, чтобы начать отслеживать новый файл, нужно его специальным образом объявить.

Подготовка файлов

В git есть концепция области подготовленных файлов. Можно представить ее как холст, на который наносят изменения, которые нужны в коммите. Сперва он пустой, но затем мы добавляем на него файлы (или части файлов, или даже одиночные строчки) командой add и, наконец, коммитим все нужное в репозиторий (создаем слепок нужного нам состояния) командой commit. В нашем случае у нас только один файл, так что добавим его:

$gitaddhello.txt

Если нам нужно добавить все, что находится в директории, мы можем использовать

$gitadd-A

Проверим статус снова, на этот раз мы должны получить другой ответ:

$git status

Onbranchmaster Initial commit

Changestobecommitted:

(use«gitrm--cached...»tounstage) new file: hello.txt

Файл готов к коммиту. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки – в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

Коммит (фиксация изменений)

Коммит представляет собой состояние репозитория в определенный момент времени. Это похоже на снапшот, к которому мы можем вернуться и увидеть состояние объектов на определенный момент времени. Чтобы зафиксировать изменения, нам нужно хотя бы одно изменение в области подготовки (мы только что создали его при помощи git add), после которого мы может коммитить:

$gitcommit-m«Initialcommit.»

Эта команда создаст новый коммит со всеми изменениями из области подготовки (добавление файла hello.txt). Ключ -m и сообщение «Initial commit.» – это созданное пользователем описание всех изменений, включенных в коммит. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии.

Удаленные репозитории

Сейчас наш коммит является локальным – существует только в директории .git на нашей файловой системе. Несмотря на то, что сам по себе локальный репозиторий полезен, в большинстве случаев мы хотим поделиться нашей работой или доставить код на сервер, где он будет выполняться. Для этого необходимо осуществить:

1. Подключение к удаленному репозиторию

Чтобы загрузить что-нибудь в удаленный репозиторий, сначала нужно к нему подключиться. В нашем руководстве мы будем использовать адрес https://github.com/tutorialzine/awesome- project, но вам посоветуем попробовать создать свой репозиторий в GitHub, BitBucket или любом другом сервисе. Регистрация и установка может занять время, но все подобные сервисы предоставляют хорошую документацию. Чтобы связать наш локальный репозиторий с репозиторием на GitHub, выполним следующую команду в терминале. Обратите внимание, что нужно обязательно изменить URI репозитория на свой.

#Thisisonlyanexample.ReplacetheURIwithyourownrepositoryad- dress.

$ git remote add origin

https://github.com/tutorialzine/awesome-

project.git

Проект может иметь несколько удаленных репозиториев одновременно. Чтобы их различать, мы дадим им разные имена. Обычно главный репозиторий называется origin.

2. Отправка изменений на сервер

Сейчас самое время переслать наш локальный коммит на сервер. Этот процесс происходит каждый раз, когда мы хотим обновить данные в удаленном репозитории. Команда, предназначенная для этого – push. Она принимает два параметра: имя удаленного репозитория (мы назвали наш origin) и ветку, в которую необходимо внести изменения (master – это ветка по умолчанию для всех репозиториев).

$ git push origin master Countingobjects:3,done.

Writingobjects:100%(3/3),212bytes| 0bytes/s,done. Total 3 (delta 0), reused 0 (delta 0)

Tohttps://github.com/tutorialzine/awesome-project.git *[newbranch]master->master

В зависимости от сервиса, который вы используете, вам может потребоваться аутентифицироваться, чтобы изменения отправились. Если все сделано правильно, то когда вы посмотрите в удаленный репозиторий при помощи браузера, вы увидите файл hello.txt

3. Клонирование репозитория

Сейчас другие пользователи GitHub могут просматривать ваш репозиторий. Они могут скачать из него данные и получить полностью работоспособную копию вашего проекта при помощи команды clone.

$gitclonehttps://github.com/tutorialzine/awesome-project.git

Новый локальный репозиторий создается автоматически с GitHub в качестве удаленного репозитория.

4. Запрос изменений с сервера

Если вы сделали изменения в вашем репозитории, другие пользователи могут скачать изменения при помощи команды pull.

$gitpulloriginmaster

Fromhttps://github.com/tutorialzine/awesome-project

*branchmaster->FETCH_HEAD Already up-to-date.

Так как новых коммитов с тех пор, как мы склонировали себе проект, не было, никаких изменений доступных для скачивания нет.

Ветвление

Во время разработки новой функциональности считается хорошей практикой работать с копией оригинального проекта, которую называют веткой. Ветви имеют свою собственную историю и изолированные друг от друга изменения до тех пор, пока вы не решаете слить изменения вместе. Это происходит по набору причин:

Рабочая стабильная версия кода сохраняется.

Различные новые функции могут разрабатываться параллельно разными программистами.

Разработчики могут работать с собственными ветками без риска, что кодовая база поменяется из-за чужих изменений.

В случае сомнений, различные реализации одной и той же идеи могут быть разработаны в разных ветках и затем сравниваться.

1. Создание новой ветки

Основная ветка в каждом репозитории называется master. Чтобы создать еще одну ветку, используем команду branch <name>

$gitbranchamazing_new_feature

Это создаст новую ветку, пока что точную копию ветки master.

2 .Переключение между ветками

Сейчас, если мы запустим branch, мы увидим две доступные опции:

$ git branch amazing_new_feature

* master

master – это активная ветка, она помечена звездочкой. Но мы хотим работать с нашей «новой потрясающей фичей», так что нам понадобится переключиться на другую ветку. Для этого воспользуемся командой checkout, она принимает один параметр – имя ветки, на которую необходимо переключиться.

$gitcheckoutamazing_new_feature

3.Слияние веток

Наша «потрясающая новая фича» будет еще одним текстовым файлом под названием feature.txt. Мы создадим его, добавим и закоммитим:

$gitaddfeature.txt

$gitcommit-m«Newfeaturecomplete.»

Изменения завершены, теперь мы можем переключиться обратно на ветку master.

$gitcheckoutmaster

Теперь, если мы откроем наш проект в файловом менеджере, мы не увидим файла feature.txt, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно воспользоваться merge для объединения веток (применения изменений из ветки

amazing_new_feature к основной версии проекта).

$gitmerge amazing_new_feature

Теперь ветка master актуальна. Ветка amazing_new_feature больше не нужна, и ее можно удалить.

$gitbranch-dawesome_new_feature

• Дополнительно

В последней части этого руководства мы расскажем о некоторых дополнительных возможностях, которые могут вам помочь.

1. Отслеживание изменений, сделанных в коммитах

У каждого коммита есть свой уникальный идентификатор в виде строки цифр и букв. Чтобы просмотреть список всех коммитов и их идентификаторов, можно использовать команду log:

Вывод gitlog

Как вы можете заметить, идентификаторы довольно длинные, но для работы с ними не обязательно копировать их целиком – первых нескольких символов будет вполне достаточно. Чтобы посмотреть, что нового появилось в коммите, мы можем воспользоваться командой show [commit]

Вывод gitshow

Чтобы увидеть разницу между двумя коммитами, используется команда diff (с указанием промежутка между коммитами):

Вывод gitdiff

Мы сравнили первый коммит с последним, чтобы увидеть все изменения, которые были когда-либо сделаны. Обычно проще использовать git difftool, так как эта команда запускает графический клиент, в котором наглядно сопоставляет все изменения.

2. Возвращение файла к предыдущему состоянию

Гит позволяет вернуть выбранный файл к состоянию на момент определенного коммита. Это делается уже знакомой нам командой checkout, которую мы ранее использовали для переключения между ветками. Но она также может быть использована для переключения между коммитами (это довольно распространенная ситуация для Гита – использование одной команды для различных, на первый взгляд, слабо связанных задач). В следующем примере мы возьмем файл hello.txt и откатим все изменения, совершенные над ним к первому коммиту. Чтобы сделать это, мы подставим в команду идентификатор нужного коммита, а также путь до файла:

$gitcheckout09bd8cc1hello.txt

3. исправление коммита

Если вы опечатались в комментарии или забыли добавить файл и заметили это сразу после того, как закоммитили изменения, вы легко можете это поправить при помощи commit – amend. Эта команда добавит все из последнего коммита в область подготовленных файлов и попытается сделать новый коммит. Это дает вам возможность поправить комментарий или добавить недостающие файлы в область подготовленных файлов. Для более сложных исправлений, например, не в последнем коммите или если вы успели отправить изменения на сервер, нужно использовать revert. Эта команда создаст коммит, отменяющий изменения, совершенные в коммите с заданным идентификатором. Самый последний коммит может быть доступен по алиасу HEAD:

$gitrevertHEAD

Для остальных будем использовать идентификаторы: $gitrevert b10cc123

При отмене старых коммитов нужно быть готовым к тому, что возникнут конфликты. Такое случается, если файл был изменен еще одним, более новым коммитом. И теперь git не может найти строчки, состояние которых нужно откатить, так как они больше не существуют.

4. Разрешение конфликтов при слиянии

Помимо сценария, описанного в предыдущем пункте, конфликты регулярно возникают при слиянии ветвей или при отправке чужого кода. Иногда конфликты исправляются автоматически, но обычно с этим приходится разбираться вручную – решать, какой код остается, а какой нужно удалить. Давайте посмотрим на примеры, где мы попытаемся слить две ветки под названием john_branchи tim_branch. Предположим, что два пользователя правят один и тот же файл: функцию, которая отображает элементы массива. Пользователь 1 использует следующий цикл:

//Useaforlooptoconsole.logcontents. for(var i=0; i<arr.length; i++) { console.log(arr[i]);

}

А Пользователь 2 использует forEach:

//UseforEachtoconsole.logcontents. arr.forEach(function(item) { console.log(item);

});

Они оба коммитят свой код в соответствующую ветку. Теперь, если они попытаются слить две ветки, они получат сообщение об ошибке:

$ git merge tim_branch Auto-mergingprint_array.js

CONFLICT(content):Mergeconflictinprint_array.js

Automaticmergefailed;fixconflictsandthencommittheresult.

Система не смогла разрешить конфликт автоматически, значит, это придется сделать разработчикам. Приложение отметило строки, содержащие конфликт:

Вывод

Над разделителем ======= мы видим последний (HEAD) коммит, а под ним – конфликтующий. Таким образом, мы можем увидеть, чем они отличаются и решать, какая версия лучше. Или вовсе написать новую. В этой ситуации мы так и поступим, перепишем все, удалив разделители, и дадим git понять, что закончили.

//Notusingforloopor forEach.

//UseArray.toString()toconsole.logcontents.

console.log(arr.toString());

Когда все готово, нужно закоммитить изменения, чтобы закончить процесс:

$gitadd-A

$gitcommit-m«Arrayprintingconflictresolved.»

Как вы можете заметить, процесс довольно утомительный и может быть очень сложным в больших проектах. Многие разработчики предпочитают использовать для разрешения конфликтов клиенты с графическим интерфейсом. (Для запуска нужно набрать git mergetool).

5. Настройка .gitignore

В большинстве проектов есть файлы или целые директории, в которые мы не хотим (и, скорее всего, не захотим) коммитить. Мы можем удостовериться, что они случайно не попадут в git add. Апри помощи файла .gitignore. Для этого:

1.Создайте вручную файл под названием .gitignore и сохраните его в директорию проекта.

2.Внутри файла перечислите названия файлов/папок, которые нужно игнорировать, каждый с новой строки.

3.Файл .gitignore должен быть добавлен, закоммичен и отправлен на сервер, как любой другой файл в проекте.

Вот хорошие примеры файлов, которые нужно игнорировать:

• Логи

• Артефакты систем сборки

• Папки node_modules в проектах node.js

• Папки, созданные IDE, например, NetbeansилиIntelliJ

• Разнообразные заметки разработчика.

Файл .gitignore, исключающий все перечисленное выше, будет выглядеть так:

*.log build/

node_modules/

.idea/ my_notes.txt

Символ слэша в конце некоторых линий означает директорию (и тот факт, что мы рекурсивно игнорируем все ее содержимое). Звездочка, как обычно, означает шаблон.

Задание

Заданием работы является создание репозитория в системе контроля версий GIT для разработки информационной системы в соответствующей предметной области из Приложения 1.

Контрольные вопросы

1. Приведите основные команды git
2. Как создать новую ветку в git?
3. Как переключиться в существующую ветку?
4. Как отправить изменения на сервер?

Практическое занятие №5. Уточнение требований: пользовательские истории и критерии приемки.

Цель работы: изучение agile-методов уточнения требований к программному обеспечению через создание пользовательских историй и критериев приемки. Результатом практической работы является отчет, содержащий примеры и анализ применения этих техник для конкретной предметной области. Отчет сдается в электронном виде (файл Word).

1. От общих требований к конкретным задачам

После анализа предметной области и выделения высокоуровневых функций системы (как описано в предыдущих занятиях), наступает этап их детализации. Традиционные технические задания часто бывают объемными, статичными и плохо адаптируются к изменениям. В гибких (agile) методологиях разработки для уточнения требований используются пользовательские истории и критерии приемки.

Пользовательская история — это краткое, простое описание одной функции системы, сформулированное с точки зрения конечного пользователя и его потребности. Её цель — не дать исчерпывающую спецификацию, а стать основой для дальнейшего обсуждения между заказчиком, аналитиком и разработчиком.

Критерии приемки — это набор конкретных, проверяемых условий, которые должны быть выполнены, чтобы пользовательская история считалась завершенной и работающей корректно. Они уточняют границы истории и определяют, что такое «рабочая функция».

2. Структура и формулировка пользовательских историй

Хорошая пользовательская история следует шаблону: «Как [роль пользователя], я хочу [действие/возможность], чтобы [цель/выгода]».

Примеры:

- *Плохо (функциональное требование):* «Реализовать форму авторизации».
- *Хорошо (пользовательская история):* «Как зарегистрированный пользователь, я хочу входить в систему с помощью логина и пароля, чтобы получить доступ к своему личному кабинету и персональным данным».

Зачем нужно «чтобы»? Эта часть истории — самая важная. Она объясняет ценность и цель функции, что помогает команде принять лучшее техническое решение и избежать реализации ненужных возможностей.

3. Создание и использование критериев приемки

Критерии приемки (Acceptance Criteria) дополняют историю и обычно формулируются в виде сценариев. Часто используется формат Given-When-Then (Дано-Когда-Тогда):

1. Дано (Given): Начальное состояние системы или контекст.
2. Когда (When): Действие, которое совершает пользователь или система.
3. Тогда (Then): Ожидаемый, проверяемый результат.

Пример для истории про авторизацию:

Критерии приемки:

1. Дано, что я нахожусь на странице входа, и у меня есть активная учетная запись.
2. Когда я ввожу корректный логин и пароль и нажимаю кнопку «Войти»,
3. Тогда система перенаправляет меня на главную страницу моего личного кабинета.
4. Дано, что я нахожусь на странице входа.
5. Когда я ввожу некорректный пароль и нажимаю «Войти»,
6. Тогда система отображает сообщение: «Неверный логин или пароль», и я остаюсь на странице входа.

4. Практическое применение: от потребности к работающей функции

Связь между понятиями, рассмотренными ранее, и новыми техниками показана на схеме:

Проблема пользователя -> Потребность -> Функция системы -> Пользовательская история -> Критерии приемки -> Детализированное требование/Тест.

Пример цепочки для системы интернет-магазина:

- Проблема: Клиенты теряются, не зная, доставлен их заказ или нет, и звонят в поддержку.
- Потребность: Возможность самостоятельно и быстро отслеживать статус заказа.
- Функция: Онлайн-трекер статуса заказов.
- Пользовательская история: «Как клиент, я хочу вводить номер своего заказа на сайте, чтобы видеть его текущий статус (например, "обработан", "отправлен", "доставлен") и номер отслеживания».
- Критерии приемки (пример одного из сценариев):
  - Дано, что на сайте есть публичная страница «Отследить заказ».
  - Когда я ввожу действительный 10-значный номер заказа и нажимаю «Найти»,
  - Тогда система отображает текущий статус заказа, дату последнего обновления и, если заказ отправлен, номер транспортной накладной.

5. Задание

Часть 1 (Анализ). используя описание предметной области из Приложения 1 (или свою собственную тему, например, «Система записи к врачу в поликлинике», «Библиотечный каталог онлайн», «Учет личных финансов»):

1. Выделите 2-3 ключевые роли пользователей (например, Пациент, Администратор, Врач).
2. Для каждой роли сформулируйте по 2 пользовательские истории, используя шаблон «Как... Я хочу... Чтобы...».
3. Для одной пользовательской истории из каждого пункта напишите 3-5 критериев приемки в формате Given-When-Then.

Часть 2 (Рефлексия). Ответьте на вопросы:

1. Как критерии приемки помогают избежать недопонимания между заказчиком и разработчиком?
2. В чем основное преимущество пользовательских историй по сравнению с обычным списком функциональных требований?
3. Всегда ли уместно использовать формат Given-When-Then для критериев приемки? Приведите пример, когда он может быть избыточным.

Контрольные вопросы:

1. Что такое пользовательская история и из каких трех основных частей она состоит?
2. Какова цель критериев приемки?
3. Опишите структуру сценария Given-When-Then.
4. Кто обычно участвует в формулировке и обсуждении пользовательских историй в agile-команде?

Практическое занятие №6. Реализация прототипа и настройка сборки.

Цель работы: изучение методов сборки и развертывания программного обеспечения для различных платформ (Windows и Linux) на примере языков PHP, C++ и C#. Понимание различий в инструментах, процессах и лучших практиках для каждого стека технологий.

1. Общие принципы сборки ПО

Сборка (build) — процесс преобразования исходного кода в исполняемый файл или готовое к развертыванию приложение. Ключевые аспекты:

- **Кросс-платформенность:** возможность собирать проект под разные ОС.
- **Зависимости:** управление библиотеками и внешними компонентами.
- **Автоматизация:** использование систем сборки для повторяемости процесса.

2. Сборка PHP-приложений

PHP — интерпретируемый язык, поэтому под «сборкой» обычно понимают **упаковку и развертывание**.

Для Windows и Linux:

1. **Composer** — менеджер зависимостей:
   - Установка: `curl -sS https://getcomposer.org/installer | php`
   - Сборка зависимостей: `composer install --no-dev`
   - Результат: папка `vendor/` с библиотеками.
2. **PHAR-архивы** (PHP Archive):
   - Создание единого исполняемого файла:

bash

```
php -d phar.readonly=0 -r "Phar::mapPhar('app.phar'); include 'phar://app.phar/index.php';"
```

- Работает на обеих платформах при наличии PHP.

3. **Docker-контейнеризация** (универсальный способ):

- `Dockerfile`:

dockerfile

```
FROM php:8.2-apache
COPY . /var/www/html
RUN composer install --no-dev
```

- Сборка: `docker build -t my-php-app .`

3. Сборка C++-приложений

C++ — компилируемый язык, требует отдельной сборки под каждую платформу.

Для Windows:

1. **Visual Studio (MSVC):**

- использование `.sln` и `.vcxproj` файлов.
- Компиляция через `MSBuild`:

bash

```
msbuild MyProject.sln /p:Configuration=Release
```

2. **MinGW / MSYS2:**

- GCC под Windows:

bash

```
g++ -o app.exe main.cpp
```

Для Linux:

1. **GCC / Clang:**

- Компиляция:

bash

```
g++ -o app main.cpp
```

2. **CMake** (кросс-платформенная система):

- `CMakeLists.txt`:

cmake

```
cmake_minimum_required(VERSION 3.10)
project(MyApp)
add_executable(app main.cpp)
```

- Сборка:

bash

```
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build .
```

3. **Кросскомпиляция** (сборка под Linux из Windows и наоборот):

- использование **MXE** (M cross environment) для сборки Windows-бинарников из Linux.
- использование **WSL** или виртуальных машин.

4. Сборка C#-приложений

C# компилируется в промежуточный язык (IL) и выполняется в среде .NET или Mono.

Для Windows (традиционно):

1. **.NET Framework (Windows-only):**

- Сборка через `MSBuild` или Visual Studio.
- исполняемый файл `.exe`, требующий .NET Framework.

2. **.NET Core / .NET 5+ (кросс-платформенный):**

- использование `dotnet CLI`:

bash

```
dotnet build -c Release
dotnet publish -c Release -r win-x64 --self-contained true
```

Для Linux:

1. **.NET Core / .NET 5+:**

- Установка SDK: `apt install dotnet-sdk`
- Сборка:

bash

```
dotnet publish -c Release -r linux-x64 --self-contained true
```

2. **Mono** (альтернативная реализация .NET):

- Компиляция: `mcs -out:app.exe main.cs`
- Запуск: `mono app.exe`

Кросс-платформенная сборка .NET:

- **Runtime Identifiers (RID):**

- win-x64, linux-x64, linux-arm

- **Конфигурация в** .csproj**:**

xml

```
<RuntimeIdentifiers>win-x64;linux-x64</RuntimeIdentifiers>
```

- **Публикация:** dotnet publish -r linux-x64

5. Автоматизация сборки: CI/CD

Для всех языков актуально использование систем непрерывной интеграции:

1. **GitHub Actions** (пример для C++):

yaml

```
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [windows-latest, ubuntu-latest]
    steps:
      - uses: actions/checkout@v4
      - run: cmake --build ./build
```

2. **Jenkins, GitLab CI, Travis CI** — аналогично.

6. Сравнительная таблица

Язык	Windows (инструменты)	Linux (инструменты)	Кросс-платформенный подход
PHP	Composer, XAMPP, Docker	Composer, Apache, Docker	Docker, PHAR
C++	MSVC, MinGW, CMake	GCC, Clang, CMake	CMake, кросскомпиляция
C#	.NET Framework, .NET 5+	.NET 5+, Mono	.NET 5+ с RID

Примеры реализации:

Вариант А: Создание готового пакета с зависимостями (Windows/Linux)

bash

```
# Шаг 1: Установка Composer (если нет)
# Windows: скачать с getcomposer.org и установить
# Linux:
sudo apt update
sudo apt install php-cli unzip
curl -sS https://getcomposer.org/installer -o composer-setup.php
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
# Шаг 2: Инициализация проекта
mkdir php-app && cd php-app
composer init --name="mycompany/myapp" --type="project"
# Шаг 3: Добавление зависимостей
composer require monolog/monolog
# Шаг 4: Создание основной структуры
mkdir src
echo "<?php require 'vendor/autoload.php'; use Monolog\Logger; echo 'Hello World!';" > src/index.php
# Шаг 5: Сборка для production (без dev-зависимостей)
composer install --no-dev --optimize-autoloader
# Результат: готовая папка для деплоя (можно архивировать)
zip -r php-app.zip src/ vendor/ composer.json
```

Вариант B: Создание PHAR-архива

```
bash
# Шаг 1: Создаем простой проект
mkdir phar-app && cd phar-app
echo "<?php echo 'Hello from PHAR!';" > index.php
# Шаг 2: Создаем stub.php для точки входа
cat > stub.php << 'EOF'
<?php
Phar::mapPhar('app.phar');
include 'phar://app.phar/index.php';
__HALT_COMPILER();
EOF
```

```
# Шаг 3: Собираем PHAR
php -d phar.readonly=0 << 'EOF'
<?php
$phar = new Phar('app.phar', 0, 'app.phar');
$phar->buildFromDirectory(__DIR__);
$phar->setStub(file_get_contents('stub.php'));
EOF
# Проверка работы
php app.phar
# Вывод: Hello from PHAR!
```

Вариант C: Docker-сборка

dockerfile

```
# Dockerfile
FROM php:8.2-apache
# Установка зависимостей системы
RUN apt-get update && apt-get install -y \
    git \
    unzip \
    && rm -rf /var/lib/apt/lists/*
# Копируем composer.json первым для кэширования
COPY composer.json /var/www/html/
# Устанавливаем Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
# Устанавливаем зависимости
RUN composer install --no-dev --optimize-autoloader
# Копируем остальной код
COPY . /var/www/html/
# Настройка Apache
RUN a2enmod rewrite
COPY .htaccess /var/www/html/.htaccess
EXPOSE 80
```

Сборка и запуск:

bash

```
docker build -t my-php-app .
docker run -p 8080:80 my-php-app
```

2. Сборка C++ приложения

Вариант A: Прямая компиляция GCC/MinGW

bash

```
# Linux (GCC):
# Шаг 1: Установка компилятора
sudo apt update
sudo apt install g++ build-essential
# Шаг 2: Создаем проект
mkdir cpp-app && cd cpp-app
cat > main.cpp << 'EOF'
#include <iostream>
#include <string>
#ifdef _WIN32
    const std::string OS = "Windows";
#else
    const std::string OS = "Linux";
#endif
int main() {
    std::cout << "Hello from " << OS << "!\n";
    return 0;
}
EOF
# Шаг 3: Компиляция
g++ -o myapp main.cpp -O2 -std=c++17
# Проверка
./myapp
```

powershell

```
# Windows (MinGW через MSYS2):
# Шаг 1: Установить MSYS2 с сайта msys2.org
```

```
# Шаг 2: Запустить MSYS2 MinGW 64-bit
# Шаг 3: Установить компилятор
pacman -S mingw-w64-x86_64-gcc
# Шаг 4: Компиляция
g++ -o myapp.exe main.cpp -O2 -std=c++17
# Шаг 5: Проверка (в обычном CMD или PowerShell)
.\myapp.exe
```

Вариант B: CMake (кросс-платформенный)

```
cmake
# CMakeLists.txt
cmake_minimum_required(VERSION 3.10)
project(MyCppApp VERSION 1.0.0)
# Настройки компилятора
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
# Добавляем исполняемый файл
add_executable(myapp src/main.cpp)
# Настройка для Windows/Linux
if(WIN32)
    target_compile_definitions(myapp PRIVATE PLATFORM_WINDOWS)
    # Добавляем ресурсы если нужно
    # set(WIN32_RESOURCE ${CMAKE_CURRENT_SOURCE_DIR}/app.rc)
    # target_sources(myapp PRIVATE ${WIN32_RESOURCE})
else()
    target_compile_definitions(myapp PRIVATE PLATFORM_LINUX)
endif()
# Установка (опционально)
install(TARGETS myapp DESTINATION bin)
bash
# Linux сборка:
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
```

```
make -j4
sudo make install  # опционально
# Windows сборка (в PowerShell с установленным Visual Studio):
mkdir build
cd build
cmake .. -G "Visual Studio 17 2022" -A x64
cmake --build . --config Release
```

Вариант C: Кросскомпиляция для Windows из Linux

```
bash
# Шаг 1: Установка MXE (M Cross Environment)
git clone https://github.com/mxe/mxe.git
cd mxe
make MXE_TARGETS=x86_64-w64-mingw32.static gcc
# Шаг 2: Добавление в PATH
export PATH=/путь/к/mxe/usr/bin:$PATH
# Шаг 3: Компиляция
x86_64-w64-mingw32.static-g++ -o app.exe main.cpp -static -O2
# Шаг 4: Проверка (нужен wine или Windows машина)
wine app.exe
```

3. Сборка C# приложения

Вариант A: .NET Core/5/6/7/8 (кросс-платформенный)

```
bash
# Шаг 1: Установка .NET SDK
# Windows: скачать с сайта Microsoft
# Ubuntu/Debian:
wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt update
sudo apt install -y dotnet-sdk-8.0
# Шаг 2: Создание проекта
mkdir cs-app && cd cs-app
```

```
dotnet new console -n MyApp
cd MyApp
# Шаг 3: Изменение Program.cs
cat > Program.cs << 'EOF'
using System;
using System.Runtime.InteropServices;
class Program {
    static void Main() {
        string os = RuntimeInformation.IsOSPlatform(OSPlatform.Windows)
            ? "Windows"
            : "Linux";
        Console.WriteLine($"Hello from {os}!");
    }
}
EOF
# Шаг 4: Сборка для текущей ОС
dotnet build -c Release
# Шаг 5: Публикация (self-contained)
# Для Windows:
dotnet publish -c Release -r win-x64 --self-contained true -p:PublishSingleFile=true
# Для Linux:
dotnet publish -c Release -r linux-x64 --self-contained true -p:PublishSingleFile=true
# Проверка
# Windows: .\bin\Release\net8.0\win-x64\publish\MyApp.exe
# Linux: ./bin/Release/net8.0/linux-x64/publish/MyApp
```

Вариант B: Мультитаргетинг для нескольких ОС

```
xml
<!-- MyApp.csproj -->
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
```

```
    <TargetFrameworks>net8.0</TargetFrameworks>
    <RuntimeIdentifiers>win-x64;linux-x64;linux-arm64</RuntimeIdentifiers>
  </PropertyGroup>
</Project>
```

Сборка всех версий:

bash

```
# Собрать все RID
dotnet publish -c Release -r win-x64
dotnet publish -c Release -r linux-x64
dotnet publish -c Release -r linux-arm64
# Или одной командой через скрипт
# build-all.sh:
#!/bin/bash
for rid in win-x64 linux-x64 linux-arm64; do
    echo "Building for $rid..."
    dotnet publish -c Release -r $rid --self-contained true
done
```

Вариант C: Сборка в Docker (мульти-архитектура)

dockerfile

```
# Dockerfile
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY . .
RUN dotnet publish -c Release -o /app/publish
# Для production
FROM mcr.microsoft.com/dotnet/runtime:8.0
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

Сборка с поддержкой нескольких архитектур:

bash

```
# Сборка для разных архитектур
```

```
docker buildx create --use
docker buildx build --platform linux/amd64,linux/arm64 -t myapp:latest .
```

4. Автоматизация через GitHub Actions

yaml

```yaml
# .github/workflows/build.yml
name: Multi-Platform Build
on: [push]
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
        include:
          - os: ubuntu-latest
            artifact_name: linux-app
            extension: ''
          - os: windows-latest
            artifact_name: windows-app
            extension: '.exe'
    steps:
    - uses: actions/checkout@v4
    # PHP сборка
    - name: PHP Build
      if: matrix.os == 'ubuntu-latest'
      run: |
        composer install --no-dev --optimize-autoloader
        zip -r php-app.zip .
# C++ сборка (CMake)
    - name: C++ Build
      run: |
        mkdir build
```

```
        cd build
        cmake .. -DCMAKE_BUILD_TYPE=Release
        cmake --build .
    # C# сборка
    - name: .NET Build
      run: |
        dotnet publish -c Release -r ${{ matrix.os == 'windows-latest' && 'win-x64' || 'linux-x64' }}
    # Загрузка артефактов
    - name: Upload Artifacts
      uses: actions/upload-artifact@v4
      with:
        name: ${{ matrix.artifact_name }}
        path: |
          build/myapp${{ matrix.extension }}
          bin/Release/**/*${{ matrix.extension }}
```

Сборка в Microsoft Visual Studio: пошаговые инструкции

1. Сборка C++ проекта в Visual Studio

Вариант A: Создание и сборка классического проекта Win32

Шаг 1: Создание проекта

1. Запустите Visual Studio 2022
2. "Создать новый проект" → "Консольное приложение C++"
3. Назовите проект WinApp и укажите расположение

Шаг 2: Настройка платформы

cpp

```cpp
// WinApp.cpp
#include <iostream>
#include <windows.h>
int main() {
    SYSTEM_INFO si;
    GetSystemInfo(&si);
```

```
    std::cout << "Hello from Windows!\n";
    std::cout << "Processor Architecture: ";

    switch (si.wProcessorArchitecture) {
        case PROCESSOR_ARCHITECTURE_AMD64:
            std::cout << "x64";
            break;
        case PROCESSOR_ARCHITECTURE_INTEL:
            std::cout << "x86";
            break;
        case PROCESSOR_ARCHITECTURE_ARM:
            std::cout << "ARM";
            break;
        default:
            std::cout << "Unknown";
    }
    std::cout << "\n";
    return 0;
}
```

Шаг 3: Настройка сборки

1. В верхней панели выберите "x64" или "x86" (платформа)
2. Выберите "Debug" или "Release" (конфигурация)
3. **Сборка → Собрать решение** (Ctrl+Shift+B)

Шаг 4: Расположение выходных файлов

- Debug x86: `WinApp\x86\Debug\WinApp.exe`
- Release x64: `WinApp\x64\Release\WinApp.exe`

Вариант B: Кросс-платформенная сборка с CMake

Шаг 1: Создание CMake проекта

1. "Создать новый проект" → "CMake проект"
2. В `CMakeLists.txt` добавьте:

```
cmake
cmake_minimum_required(VERSION 3.10)
```

```
project(CrossPlatformApp)
# Настройки для Windows
if(WIN32)
    add_executable(${PROJECT_NAME} WIN32 src/main.cpp)
    target_compile_definitions(${PROJECT_NAME} PRIVATE _WINDOWS)
else()
    add_executable(${PROJECT_NAME} src/main.cpp)
endif()
# Общие настройки
target_compile_features(${PROJECT_NAME} PRIVATE cxx_std_17)
```

Шаг 2: Настройка конфигураций CMake

1. Откройте CMakeSettings.json
2. Добавьте конфигурации:

json

```
{
  "configurations": [
    {
      "name": "x64-Debug",
      "generator": "Visual Studio 17 2022",
      "configurationType": "Debug",
      "inheritEnvironments": [ "msvc_x64_x64" ],
      "buildRoot": "${projectDir}\\out\\build\\${name}",
      "installRoot": "${projectDir}\\out\\install\\${name}"
    },
    {
      "name": "Linux-Debug",
      "generator": "Unix Makefiles",
      "configurationType": "Debug",
      "remoteCMakeListsRoot": "/home/user/project",
      "remoteBuildRoot": "/home/user/project/out/build/${name}",
      "remoteInstallRoot": "/home/user/project/out/install/${name}"
    }
```

```
  ]
}
```

Шаг 3: Сборка для Windows

1. Выберите конфигурацию "x64-Debug"
2. **Сборка → Собрать все** (F7)

Шаг 4: Сборка для Linux (через WSL)

1. Установите расширение "C++ для Linux"
2. Подключитесь к WSL: **Сервис → Параметры → Кроссплатформенный → Диспетчер подключений**
3. Выберите конфигурацию "Linux-Debug"
4. Соберите проект

2. Сборка C# проекта в Visual Studio

Вариант A: Классическое .NET приложение

Шаг 1: Создание проекта

1. "Создать новый проект" → "Консольное приложение C#"
2. Выберите .NET 8.0

Шаг 2: Настройка для нескольких платформ

xml

```
<!-- Измените файл проекта (.csproj) -->
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <!-- Включаем поддержку нескольких RID -->
    <RuntimeIdentifiers>win-x64;linux-x64;linux-arm64</RuntimeIdentifiers>
    <!-- Включаем AOT компиляцию (опционально) -->
    <PublishAot>true</PublishAot>
  </PropertyGroup>
</Project>
```

Шаг 3: Публикация для Windows x64

1. Щелкните правой кнопкой на проекте → **Опубликовать**
2. Выберите "Новый профиль" → "Папка"

3. Настройки публикации:
   - Конфигурация: Release
   - Целевая среда выполнения: win-x64
   - Режим развертывания: автономное приложение
4. Нажмите "Опубликовать"

Шаг 4: Публикация через командную строку в VS

powershell

```
# Откройте терминал Package Manager в VS
dotnet publish -c Release -r win-x64 --self-contained true -p:PublishSingleFile=true
dotnet publish -c Release -r linux-x64 --self-contained true -p:PublishSingleFile=true
```

Вариант B: Создание установщика (Windows)

Шаг 1: Добавление проекта установки

1. Правой кнопкой на решение → **Добавить** → **Новый проект**
2. Найти "Setup Project" или использовать "WiX Toolset"

Шаг 2: WiX Toolset установка

1. Установите расширение "WiX Toolset Visual Studio Extension"
2. Создайте новый проект "WiX Project"
3. В Product.wxs настройте установку:

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
<Product Id="*" Name="MyApp" Language="1033"
         Version="1.0.0.0" Manufacturer="MyCompany">
<Package InstallerVersion="200" Compressed="yes" />

  <MajorUpgrade DowngradeErrorMessage="A newer version is already installed." />

  <Feature Id="ProductFeature" Title="MyApp" Level="1">
    <ComponentGroupRef Id="ProductComponents" />
  </Feature>
```

```
  </Product>

  <Fragment>
    <Directory Id="TARGETDIR" Name="SourceDir">
      <Directory Id="ProgramFilesFolder">
        <Directory Id="INSTALLFOLDER" Name="MyApp" />
      </Directory>
    </Directory>
  </Fragment>

  <Fragment>
    <ComponentGroup Id="ProductComponents" Directory="INSTALLFOLDER">
      <Component Id="MainExecutable">
        <File Source="$(var.MyApp.TargetPath)" />
      </Component>
    </ComponentGroup>
  </Fragment>
</Wix>
```

Шаг 3: Сборка установщика

1. Добавьте ссылку на основной проект
2. **Сборка → Собрать решение**

3. Сборка PHP проекта через Visual Studio

Вариант A: использование PHP Tools for Visual Studio

Шаг 1: Установка расширения

1. **Расширения → Управление расширениями**
2. Найдите "PHP Tools for Visual Studio"
3. Установите и перезапустите VS

Шаг 2: Создание PHP проекта

1. "Создать новый проект" → "PHP Project"
2. В структуре проекта:

text

MyPhpApp/

```
├── src/
│   └── index.php
├── tests/
├── composer.json
└── .htaccess
```

Шаг 3: Настройка Composer

1. Правой кнопкой на проект → **PHP Tools** → **Composer** → **Initialize Composer**
2. Добавьте зависимости через интерфейс:
   - Правой кнопкой на проект → **Управление пакетами Composer**

Шаг 4: Сборка и публикация

1. Создайте скрипт сборки (`build.ps1`):

powershell

```powershell
# build.ps1
param([string]$Configuration = "Release")
if ($Configuration -eq "Release") {
    composer install --no-dev --optimize-autoloader
    Remove-Item -Recurse -Force -Path "vendor\composer\installed.php"
} else {
    composer install
}
# Создание ZIP архива
$version = Get-Date -Format "yyyyMMdd-HHmm"
$archive = "MyPhpApp-$version.zip"
Compress-Archive -Path "src\", "vendor\", "composer.json", "index.php" -DestinationPath $archive
Write-Host "Created: $archive"
```

2. Добавьте задачу сборки в проект:

xml

```xml
<!-- Добавьте в файл проекта .phpproj если используется -->
<PropertyGroup>
```

<PostBuildEvent>powershell -ExecutionPolicy Bypass -File "$(ProjectDir)build.ps1" -Configuration $(ConfigurationName)</PostBuildEvent>
</PropertyGroup>

4. Автоматизация сборки в VS Team Services (Azure DevOps)

Настройка pipeline для C++

Шаг 1: Создание YAML pipeline

yaml

```yaml
# azure-pipelines.yml
trigger:
- main
pool:
  vmImage: 'windows-latest'
variables:
  solution: '**/*.sln'
  buildPlatform: 'x64'
  buildConfiguration: 'Release'
stages:
- stage: BuildWindows
  jobs:
  - job: Build
    steps:
    - task: VSBuild@1
      inputs:
        solution: '$(solution)'
        platform: '$(buildPlatform)'
        configuration: '$(buildConfiguration)'

    - task: VSTest@2
      inputs:
        platform: '$(buildPlatform)'
        configuration: '$(buildConfiguration)'
```

```
    - task: PublishBuildArtifacts@1
      inputs:
        PathtoPublish: '$(Build.SourcesDirectory)/$(BuildConfiguration)'
        ArtifactName: 'Windows-Build'
- stage: BuildLinux
  dependsOn: BuildWindows
  pool:
    vmImage: 'ubuntu-latest'
  jobs:
  - job: BuildLinux
    steps:
    - script: |
        mkdir build
        cd build
        cmake ..
        make
      displayName: 'Build on Linux'
```

Настройка pipeline для .NET

```
yaml
# azure-pipelines-dotnet.yml
trigger:
- main
variables:
  buildConfiguration: 'Release'
jobs:
- job: BuildMultiPlatform
  strategy:
    matrix:
      windows_x64:
        runtime: 'win-x64'
        os: 'windows-latest'
      linux_x64:
```

```
      runtime: 'linux-x64'
      os: 'ubuntu-latest'
    linux_arm64:
      runtime: 'linux-arm64'
      os: 'ubuntu-latest'
  pool:
    vmImage: $(os)

  steps:
  - task: UseDotNet@2
    inputs:
      packageType: 'sdk'
      version: '8.x'

  - script: |
      dotnet restore
      dotnet build -c $(buildConfiguration) -r $(runtime)
      dotnet publish -c $(buildConfiguration) -r $(runtime) --self-contained true
    displayName: 'Build and Publish'
```

5. Полезные команды в VS Developer Command Prompt

```
cmd
:: Откройте Developer Command Prompt for VS 2022
:: Сборка C++ проекта
msbuild MyProject.sln /p:Configuration=Release /p:Platform=x64 /t:rebuild
:: Публикация .NET проекта
dotnet publish MyApp.csproj -c Release -r win-x64
:: Создание пакета NuGet
nuget pack MyPackage.nuspec
:: Анализ зависимостей
dumpbin /DEPENDENTS MyApp.exe
:: Просмотр манифеста
mt.exe -nologo -manifest MyApp.exe.manifest -outputresource:MyApp.exe;1
```

6. Шаблоны конфигураций для разных сценариев

Шаблон 1: Многоязычное решение (C++/C#)

text

```
Solution/
├── NativeCore/              # C++ библиотека
│   ├── NativeCore.vcxproj
│   └── NativeCore.h
├── ManagedWrapper/          # C# обертка
│   ├── ManagedWrapper.csproj
│   └── NativeInterop.cs
├── WebInterface/            # PHP/Web API
│   ├── index.php
│   └── api/
└── Installer/               # Установщик
    ├── Installer.wixproj
    └── Product.wxs
```

Шаблон 2: BuildConfiguration.props (общие настройки)

xml

```
<!-- Directory.Build.props -->
<Project>
  <PropertyGroup>
    <Version>1.0.0</Version>
    <Copyright>© 2024 MyCompany</Copyright>
    <OutputPath>$(SolutionDir)bin\$(Platform)\$(Configuration)\</OutputPath>
  </PropertyGroup>
  <!-- Настройки для Debug -->
  <PropertyGroup Condition="'$(Configuration)'=='Debug'">
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <Optimize>false</Optimize>
  </PropertyGroup>

  <!-- Настройки для Release -->
```

```
  <PropertyGroup Condition="'$(Configuration)'=='Release'">
    <DefineConstants>RELEASE</DefineConstants>
    <Optimize>true</Optimize>
  </PropertyGroup>
</Project>
```

Задание 1: Создайте многоязычное решение:

1. C++ DLL с математическими функциями
2. C# консольное приложение, использующее эту DLL через P/Invoke
3. PHP веб-страницу, вызывающую C# приложение

Задание 2: Настройте автоматическую сборку:

1. При коммите в main ветку
2. Собираются все конфигурации (Debug/Release x86/x64)
3. Запускаются тесты
4. Создаются установочные пакеты
5. Делается деплой на тестовый сервер

Решение:

powershell

```
# Скрипт полной сборки FullBuild.ps1
param([string]$BuildNumber = "1.0.0")
Write-Host "=== Полная сборка решения ===" -ForegroundColor Green
# 1. Сборка C++ проекта
Write-Host "1. Сборка NativeMath..." -ForegroundColor Yellow
msbuild NativeMath.sln /p:Configuration=Release /p:Platform=x64 /p:BuildNumber=$BuildNumber
# 2. Сборка C# проекта
Write-Host "2. Сборка MathWrapper..." -ForegroundColor Yellow
dotnet build MathWrapper.csproj -c Release -r win-x64
# 3. Сборка PHP компонентов
Write-Host "3. Подготовка PHP компонентов..." -ForegroundColor Yellow
composer install --no-dev --optimize-autoloader
# 4. Создание пакета
Write-Host "4. Создание пакета..." -ForegroundColor Yellow
```

```
$packageName = "MathSuite-$BuildNumber.zip"
Compress-Archive -Path @("bin/Release/x64/", "php-web/", "README.md") -DestinationPath $packageName
Write-Host "Готово! Пакет: $packageName" -ForegroundColor Green
```

Контрольный список перед сборкой в VS

1. **Проверка зависимостей:**
   - Правой кнопкой на решение → **Анализ зависимостей**
   - Убедитесь, что все NuGet пакеты актуальны
2. **Настройки компилятора:**
   - Для C++: **Свойства проекта → C/C++ → Оптимизация**
   - Для C#: **Свойства проекта → Сборка → Дополнительно**
3. **Подписывание сборок:**
   - **Свойства проекта → Подписывание**
   - Выберите файл ключа (.snk или .pfx)
4. **Версионирование:**
   - используйте `[assembly: AssemblyVersion("1.0.*")]` в C#
   - Или настройте в `Directory.Build.props`
5. **Тестирование:**
   - Запустите все тесты перед финальной сборкой
   - **Тест → Запустить все тесты** (Ctrl+R, A)

Такая настройка позволяет эффективно использовать возможности Visual Studio для сборки проектов любого типа и сложности.

Контрольные вопросы:

1. В чем основное отличие сборки интерпретируемых (PHP) и компилируемых (C++) языков?
2. Какой инструмент позволяет единообразно собирать C++ проект под Windows и Linux?
3. Что такое Runtime Identifier (RID) в .NET и для чего он используется?
4. Как Docker помогает решить проблему кросс-платформенной сборки и развертывания?

Заключение

Настройка сборки — критически важный этап в разработке ПО, обеспечивающий переносимость, повторяемость и надежность развертывания. Выбор инструментов зависит от языка, целевых платформ и требований проекта. Современные подходы (CMake, .NET CLI, Docker) значительно упрощают создание кросс-платформенных сборок.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

1. Рудаков А.В. Технология разработки программных продуктов: учебное издание / Рудаков А.В. - Москва : Академия, 2024. - 208 с. (Специальности среднего профессионального образования). - URL: https://academia-moscow.ru - Режим доступа: Электронная библиотека «Academiamoscow». - Текст : электронный.

Дополнительная литература

1. Казанский, А. А. Объектно-ориентированное программирование. Visual Basic : учебник для среднего профессионального образования / А. А. Казанский. — 2-е изд. — Москва : Издательство Юрайт, 2025. — 295 с. — (Профессиональное образование). — ISBN 978-5-534-21384-3. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/569868.

2. Казанский, А. А. Программирование на visual c# 2013.: учебное пособие для СПО / Казанский А. А.. – Москва : Юрайт, 2020. – 191 с. – ISBN 978-5-534-02721-1. – URL: https://urait.ru/book/programmirovaniena-visual-c-2013-452454. – Текст : электронный.

ПРИЛОЖЕНИЕ 1. Предметная область для анализа и ее описание

№	Предметная область	Описание предметной области
1	Предметная область «операции с недвижимостью»	Администрация агентства недвижимости заказала разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о квартирах, которые покупает и продает агентство, расценках на квартиры, расценках на оказываемые услуги, о покупателях и совершенных сделках. Система должна выдавать отчеты по запросу менеджера: прайс-лист на квартиры, на услуги.
2	Предметная область «медицинские услуги»	Руководство частной медицинской клиники заказало разработку информационной системы для административной группы. Система предназначена для обработки данных о клиентах, врачах, о перечне медицинских услуг (с расценками и описанием). Система должна выдавать информацию по запросу менеджера клиники: талон на посещение, прайс- листы по услугам.
3	Предметная область управляющего рекламным агентством	Руководство рекламного агентства заказало разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о клиентах, о продукции, предоставляемых услугах, стоимости пакета заказываемой рекламы и медиа-план для заказчика. Система должна выдавать отчеты по запросу менеджера: перечень изготавливаемо рекламной продукции со стоимостью, квитанция для расчета.

№	Предметная область	Описание предметной области
4	Система учета заказов и их выполнение в мебельном салоне	Администрация компании по производству и продаже мебели, заказала разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о клиентах, о товарах (характеристика товара, возможный материал изготовления), услугах, о учете заказов. Система должна выдавать отчеты по запросу менеджера: прайс-лист на оказываемые услуги, бланк расчета.
5	Разработка автоматизированной системы заказов по каталогу	Администрация торговой компании заказала разработку информационной системы заказов товаров по каталогам. Система предназначена для обработки данных о клиентах, товарах в каталогах, сроках поставок и дополнительных услугах, оказываемых фирмой. Система должна выдавать отчеты по запросу менеджера: прайс-лист перечень товаров со стоимостью (по видам товара), квитанция для расчета.
6	Предметная область продавца-консультанта магазина «оптика»	Администрация магазина «оптика» заказала разработку ИС для отдела работы с покупателем. Система предназначена для обработки данных о клиенте, о материалах, учет заказов. Система должна выдавать отчеты по запросу продавца-консультанта магазина: расчеты с клиентами, прайс-лист на услуги.
7	Предметная область «расписание для спорткомплекса»	Администрация спорткомплекса заказала разработку ИС для организации своей работы. Система предназначена для обработки данных о кол-ве человек в группе, виде занятий, учет помещений, фамилии тренеров. Система должна выдавать отчеты по запросу менеджера спорткомплекса: отчеты по загрузкам тренера и помещений.

№	Предметная область	Описание предметной области
8	Предметная область администратора ресторана	Администрация ресторана заказала разработку ИС. Система предназначена для обработки данных о местах и площадях залов, информация о заказах на места, предварительный заказ блюд. Система должна выдавать отчеты по запросу администратора ресторана: бланк счета.
9	Система организации чемпионата по определенному виду спорта	Администрация города заказала разработку ИС для спортивного комитета. Система предназначена для обработки данных о стадионах, о спортсменах, тренерах, а также о времени проведения игр. Система должна выдавать отчеты по запросу члена комитета: расписание игр.
10	Расчеты с поставщиками	Система должна содержать информацию о расчете с поставщиками продукции включая данные: о дате поставки и о самом поставщике, а также информацию о поставляемых изделиях. Выходная информация: документы по расчету с поставщиками, перечень имеющихся в наличии изделий.
11	Предметная область менеджера автосервиса	Администрация службы автосервиса заказала разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о комплектующих, о заказах на комплектующие, расценках по оказываемым услугам, о машинах и их обслуживании. Система должна выдавать отчеты по запросу менеджера автосервиса: прайс-лист на оказываемые услуги, документы по заказам, квитанции по оплате услуг и т.д.

№	Предметная область	Описание предметной области
12	Предметная область «страхование населения»	Руководство страховой компании заказало разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о видах страховок, их стоимость, о совершенных сделках, о клиентах, сроках действия страховки. Система должна выдавать отчеты по запросу менеджера: прайс-лист по видам страховки, бланк страхования, информация о клиентах и т.д.
13	Предметная область управляющего рекламным агентством	Руководство рекламного агентства заказало разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о клиентах, о продукции, предоставляемых услугах, стоимости пакета заказываемой рекламы и медиа-план для заказчика. Система должна выдавать отчеты по запросу менеджера: перечень изготавливаемо рекламной продукции со стоимостью (по видам продукции), квитанция для расчета, медиа-план,стоимость услуг и т.п.
14	Предметная область оператора агентства по трудоустройству	Администрация агентства по трудоустройству заказала разработку информационной системы для отдела по работе с клиентами. Система предназначена для обработки данных о специалистах, стоящих на учете, фирмах, где требуются специалисты, и требованиях, которые к специалистам предъявляются. Кроме того в системе должны обрабатываться данные об услугах предоставляемых агентством. Система должна выдавать отчеты по запросу менеджера: бланк анкеты, список вакансий по разделам, бланк направления на работу и прочие необходимые справки.

№	Предметная область	Описание предметной области
15	Система исследования товарного рынка (товар на выбор)	Администрация предприятия заказала разработку информационной системы для отдела маркетинга. Система предназначена для обработки данных о продажах товара за определенный промежуток времени (по подразделениям), ценах на этот же товар у конкурентов, статистике об альтернативных товарах, взаимозаменяющих элементах и т.п.. Система должна выдавать отчеты по запросу менеджера: отчет о динамике продаж с графическим анализом, отчет о движении товара, отчет о состоянии рынка и т.д.
16	Система учета заказов и их выполнение в строительной фирме (ремонт квартир)	Администрация строительной компании, занимающейся ремонтом квартир, заказала разработку информационной системы для отдела работы с клиентами. Система предназначена для обработки данных о клиентах и перечне услуг, а также учете заказов, используемом материале и учет затрат по заказам. Кроме того, в системе должна храниться база фотографий с образцами ремонта и в целом отремонтированных квартир. Система должна выдавать отчеты по запросу менеджера: прайс-лист на оказываемые услуги, бланк расчета и другие документы необходимые для работы компании с клиентами.
17	Предметная область оператора отделения связи (подписка на издания)	Руководство отделения связи федеральной почтовой службы заказало разработку информационной системы для отдела оформления подписки на периодические издания. Система предназначена для обработки данных о клиентах, изданиях, каталогах со стоимостью подписки (по разделам и тематике), а также услугах, оказываемых подписчикам. Система должна выдавать отчеты по запросу менеджера: прайс-лист на оказываемые услуги, квитанция подписки, а также другие документы необходимые в процессе работы.

№	Предметная область	Описание предметной области
18	Предметная область «система подсчета голосов в избирательных компаниях»	Администрация города заказала разработку ИС для избиркома. Система предназначена для обработки данных об избирателях, о кандидатах, информация об избирательных участках. Система должна выдавать отчеты по запросу члена комиссии: бланк голосования, формирование итоговых протоколов по участкам, округам и городу. Ведомость учета избирателей.
19	Предметная область администратора ателье мод	Администрация ателье мод заказала разработку ИС. Система предназначена для обработки данных о клиентах, сроки выполнения заказов, информация об исполнителях, перечень услуг и их стоимость, учет затрат и заказов. Система должна выдавать отчеты по запросу администратора ателье мод: прайс-лист на услуги, квитанция для расчета с клиентами, отчеты по запросам.
20	Обработка оборотных ведомостей	Система должна хранить и обновлять оборотные ведомости по материалам. Для различных материалов содержатся данные о цене, количестве и сумме. По цене и количеству необходимо иметь остатки на начало года или месяца, поступления от подотчетного лица и с центрального склада, выдачи в производство и остатки на конец года. Выходная информация: оборотные ведомости по месяцам.

№	Предметная область	Описание предметной области
21	Предметная область бухгалтера расчетчика (задача начисления з/платы)	Система должна содержать информацию об учете заработной платы сотрудников предприятия. Для каждого лица в базе должны содержаться данные о профессии, должности, начислениях заработной платы, премиях, начислениях по больничному листу, задолженностям по выплатам на начало месяца. система должна содержать для каждого сотрудника информацию об удержании, включая налоги, алименты и сумму к выдаче. Выходная информация: ведомость на получение з/платы, расчетные листки, бухгалтерские справки по доходам и расходам.
22	Предметная область оператора кинотеатра (на примере кинотеатра «иллюзион»)	Администрация кинотеатра заказала разработку ИС. Система предназначена для обработки данных о времени проведения сеансов, включая названия фильмов, о количестве мест в зале, о ценах. Система должна выдавать отчеты по запросу оператора кинотеатра: расписание сеансов со стоимость билетов, билет на определенный сеанс.
23	Предметная область «Расчет стоимости ПК из комплектующих»	Администрация магазина заказала разработку ИС. Система предназначена для расчета суммарной стоимости компьютера при известных ценах на комплектующие. Система должна обеспечивать продавцу выбор комплектующих из списка (с известными ценами). При расчете итоговой стоимости должна учитываться стоимость доставки (если она необходима). Так же система должна запрашивать ФИО и адрес клиента. Система должна выводить квитанцию на оплату с указанием ФИО, адреса клиента, перечнем всех комплектующих и итоговой ценой.

№	Предметная область	Описание предметной области
24	Предметная область «Организация мероприятий и конференций»	Компания-организатор мероприятий заказала разработку ИС для автоматизации процесса подготовки и проведения конференций. Система должна позволять администраторам создавать мероприятия, устанавливать даты, определять бюджет и максимальное количество участников. Участники должны иметь возможность регистрироваться на мероприятие, выбирать интересующие их секции и мастер-классы из предложенного расписания. Система должна автоматически формировать индивидуальные программы для каждого участника, учитывать диетические предпочтения при формировании заявок на кейтеринг, а также генерировать посадочные талоны и бейджи. Организаторам необходим функционал для анализа посещаемости секций и формирования финансовых отчетов по мероприятию.
25	Предметная область «Учет и обслуживание книжного фонда библиотеки»	Центральная городская библиотека заказала разработку ИС для учета книжного фонда и автоматизации процессов обслуживания читателей. Система должна обеспечивать каталогизацию новых поступлений с возможностью поиска по автору, названию, жанру и ключевым словам. Читатели должны иметь возможность бронировать книги онлайн, продлевать срок пользования и просматривать историю взятых книг. Для библиотекарей система должна предусматривать учет выданных и возвращенных книг, формирование напоминаний о просроченных возвратах, автоматическое резервирование книг в очередь на популярные издания. Дополнительно система должна формировать статистические отчеты по востребованности книг и читательской активности.

№	Предметная область	Описание предметной области
26	Предметная область «Управление задачами в IT-проектах»	IT-компания заказала разработку внутренней ИС для управления задачами в рамках проектов разработки ПО. Система должна позволять менеджерам проектов создавать проекты, разбивать их на задачи, назначать исполнителей и устанавливать сроки выполнения. Разработчики должны иметь возможность менять статус задач, добавлять комментарии, прикреплять файлы и вести учет затраченного времени. Система должна автоматически строить диаграммы Ганта, вычислять загрузку сотрудников, формировать отчеты о прогрессе проекта и отправлять уведомления о приближающихся дедлайнах. Для руководителей должен быть доступен дашборд с ключевыми метриками проектов.
27	Предметная область «Мониторинг состояния здоровья пациентов»	Медицинский центр заказал разработку ИС для дистанционного мониторинга состояния пациентов с хроническими заболеваниями. Система должна получать данные с медицинских устройств пациентов (глюкометры, тонометры, фитнес-трекеры) и сохранять их в электронных медицинских картах. Врачи должны иметь доступ к истории показателей, устанавливать целевые диапазоны и получать автоматические оповещения при выходе показателей за пределы нормы. Пациенты должны видеть графики своих показателей, получать рекомендации по образу жизни и напоминания о приеме лекарств. Система также должна формировать отчеты для лечащих врачей перед плановыми приемами.

№	Предметная область	Описание предметной области
28	Предметная область «Учет аренды помещений в бизнес-центре»	Управляющая компания бизнес-центра заказала разработку ИС для учета аренды офисных помещений. Система должна вести базу данных арендаторов с историей платежей, автоматически формировать счета на оплату аренды и коммунальных услуг, отслеживать просроченные платежи и начислять пени. Дополнительно система должна управлять бронированием переговорных комнат и общих зон, а также формировать отчеты по загрузке помещений и доходности бизнес-центра.
29	Предметная область «Подбор персонала и управние вакансиями»	Кадровое агентство заказало разработку ИС для автоматизации процесса подбора персонала. Система должна позволять размещать вакансии на различных платформах, собирать и структурировать резюме кандидатов, автоматически определять соответствие кандидатов требованиям вакансий. Рекрутеры должны иметь возможность планировать собеседования, вести заметки по кандидатам и формировать предложения о работе. Система также должна строить аналитику по эффективности источников найма и времени закрытия вакансий.
30	Своя область	Требования к описанию предметной области приведены ниже

Шаблон для формулирования требований к любой предметной области:

1. **Основные пользователи и их роли**
2. **Ключевые бизнес-процессы**
3. **Требуемые функции системы**
4. **Входные данные системы**
5. **Выходные данные/отчеты**
6. **Интеграции с внешними системами**
7. **Требования к безопасности**
8. **Ограничения и условия**

Пример заполнения для области «Расчет стоимости ПК из комплектующих»:

1. **Пользователи:** Продавец, Администратор, Клиент
2. **Процессы:** Подбор комплектующих → Расчет стоимости → Оформление заказа → Формирование квитанции
3. **Функции:** Выбор из каталога, расчет итога, добавление доставки, ввод данных клиента
4. **Входные данные:** Цены на комплектующие, выбор пользователя, ФИО и адрес клиента
5. **Выходные данные:** Квитанция с детализацией и итоговой суммой
6. **Интеграции:** База данных товаров, система печати
7. **Безопасность:** Доступ по ролям, защита данных клиентов
8. **Ограничения:** Поддержка только наличного расчета, работа в offline-режиме

Составитель
Витвицкий Максим Николаевич

Методические указания по выполнению самостоятельной работы
для студентов очной формы обучения
по направлению специальности
09.02.07 «Информационные системы и программирование»

Публикуется в авторской редакции