# Практическая работа

**Построение архитектуры программного средства.**

**Цель:** Изучение базовых принципов и шаблонов построения архитектуры приложений, выбор стратегии и шаблона проектирования, которые помогут при проектировании слоев, компонентов и сервисов решения, а также визуальное проектирование архитектуры приложения с использованием Microsoft Visio.

**Задание:**

С целью реализации начальных этапов разработки ПС в соответствии с техническим заданием создать архитектуру приложения.

***Теоретические сведения***

**Построение архитектуры программного средства**

При возникновении потребностей в заказе, приобретении, разработке, эксплуатации и сопровождении программ перед всеми сторонами, вовлеченными в жизненный цикл программного средства (ПС), возникает целый ряд вопросов, связанных с определением и детальным структурированием жизненного цикла (ЖЦ) ПС, с организационными и техническими правами и обязанностями сторон, с управлением ЖЦ и контролем за его реализацией. Одним из действенных инструментов для решения данных вопросов является использование унифицированных подходов, закрепленных в современных международных и российских стандартах.

Понятия «жизненный цикл системы» или «жизненный цикл программного средства» часто появляются в статьях и звучат в разговорах разработчиков, по крайней мере, руководителей проектов и подразделений.

Всем понятно, что относятся они к тому, что и в какой последовательности должно делаться при создании и эксплуатации систем. Но прежде чем две организации или два специалиста договорятся о том, что конкретно входит или не входит в ЖЦ, проходит значительное время. А позже вполне может обнаружиться, что эти двое (две «стороны») все-таки по-разному понимают, какие работы будут входить в ЖЦ, а какие - нет, какие проверки будут планироваться, когда и т. д. Естественно, общие принципы организации работ описаны давно, но что делать сторонам в конкретном проекте — это каждый раз приходится решать заново.

В стандартах, регламентирующих жизненный цикл программных средств, обобщаются опыт и результаты исследований множества специалистов и рекомендуются наиболее эффективные современные методы и процессы создания и развития комплексов программ. В результате таких обобщений оттачиваются технологические процессы и приемы разработки, а также методическая база для их автоматизации.

ЖЦ ПС в стандартах представляет собой набор этапов, частных работ и операций в последовательности их выполнения и взаимосвязи, регламентирующих ведение работ от подготовки технического задания до завершения испытаний ряда версий и окончания эксплуатации ПС или информационной системы (ИС).

Стандарты включают правила описания исходной информации, способов и методов выполнения операций, устанавливают правила контроля технологических процессов, требования к оформлению их результатов, а также регламентируют содержание технологических и эксплуатационных документов на комплексы программ. Они определяют организационную структуру коллектива, обеспечивают распределение и планирование заданий, а также, контроль за ходом создания ПС.

Для того чтобы привнести порядок и понимание, общие для любых сторон, участвующих в ЖЦ систем и ПС, давно разрабатывались стандарты различных уровней утверждения - национальные и международные.

В России основы построения и использования профилей стандартов ЖЦ ПС заложены принятием в качестве базового стандарта ГОСТ Р ИСО/МЭК 12207. Данный документ введен в действие с 1 июля 2000 г., тесно взаимоувязан с рядом стандартов, принятых ранее, и с некоторыми стандартами, разрабатываемыми в данное время на основе прямого применения стандартов ИСО.

Актуальность стандарта ГОСТ Р ИСО/МЭК 12207 для современных условий настолько высока, что принятие в ISO его исходного, международного варианта вскоре вызвало самую положительную оценку российских экспертов. Был дан ряд рекомендаций по его использованию в реальных условиях.

В данном стандарте программное обеспечение (ПО) или программный продукт определяется как набор компьютерных программ, процедур и связанной с ними документации и данных.

Процесс определяется как совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

В соответствии с ГОСТ Р ИСО/МЭК 12207 все процессы ЖЦ ПО разделены на три группы:

1) Основные процессы:

−приобретение;

−поставка;

−разработка;

−эксплуатация;

−сопровождение.

2) Вспомогательные процессы:

−документирование;

−управление конфигурацией;

−обеспечение качества;

−верификация;

−аттестация;

−совместная оценка;

−аудит;

−разрешение проблем.

3) Организационные процессы:

−управление;

−усовершенствование;

−создание инфраструктуры;

−обучение.

Процесс разработки предусматривает действия и задачи, выполняемые разработчиком, и включает следующие действия:

А) Подготовительная работа, которая начинается с выбора модели ЖЦ ПО, соответствующей масштабу, значимости и сложности проекта. Действия и задачи процесса должны соответствовать выбранной модели. Разработчик должен выбрать, адаптировать к условиям проекта и использовать согласованные с заказчиком стандарты, методы и средства разработки, а также составить план выполнения работ.

Б) Анализ требований к системе подразумевает определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т.д. Требования к системе оцениваются исходя из критериев реализуемости и возможности проверки при тестировании.

Анализ требований к ПО предполагает определение следующих характеристик для каждого компонента ПО:

−функциональных возможностей, включая характеристики производительности и среды функционирования компонента;

−внешних интерфейсов;

−спецификаций надежности и безопасности;

−эргономических требований;

−требований к используемым данным;

−требований к установке и приемке;

−требований к пользовательской документации;

−требований к эксплуатации и сопровождению.

Требования к ПО оцениваются исходя из критериев соответствия требованиям к системе, реализуемости и возможности проверки при тестировании.

В) Проектирование архитектуры системы на высоком уровне заключается в определении компонентов ее оборудования, ПО и операций, выполняемых эксплуатирующим систему персоналом. Архитектура системы должна соответствовать требованиям, предъявляемым к системе, а также принятым проектным стандартам и методам.

Проектирование архитектуры ПО включает следующие задачи:

− трансформацию требований к ПО в архитектуру, определяющую на высоком уровне структуру ПО и состав ее компонентов;

− разработку и документирование программных интерфейсов ПО и баз данных;

− разработку предварительной версии пользовательской документации;

− разработку и документирование предварительных требований к тестам и планам интеграции ПО.

Архитектура компонентов ПО должна соответствовать требованиям, предъявляемым к ним, а также принятым проектным стандартам и методам.

Г) Детальное проектирование ПО включает следующие задачи:

− описание компонентов и интерфейсов между ними на более низком уровне, достаточном для их последующего самостоятельного кодирования и тестирования;

− разработку и документирование детального проекта базы данных;

− обновление (при необходимости) пользовательской документации;

Д) Кодирование и тестирование ПО охватывает задачи:

− разработку и документирование каждого компонента ПО и базы данных, а также совокупности тестовых процедур и данных для их тестирования;

− тестирование каждого компонента ПО и базы данных на соответствие предъявляемых к ним требованиям. Результаты тестирования компонентов должны быть документированы;

− обновление (при необходимости) пользовательской документации;

− обновление плана интеграции ПО.

Е) Интеграция ПО предусматривает сборку разработанных компонентов ПО в соответствии с планом интеграции и тестирование агрегированных компонентов. Для каждого из агрегированных компонентов разрабатываются наборы тестов и тестовые процедуры, предназначенные для проверки каждого из квалификационных требований при последующем квалификационном тестировании.

Ж) Квалификационное тестирование - это набор критериев и условий,

которые необходимо выполнить, чтобы квалифицировать программный продукт как соответствующий своим спецификациям и готовый к использованию в условиях эксплуатации.

Квалификационное тестирование ПО проводится разработчиком, в присутствии заказчика (по возможности), для демонстрации того, что ПО удовлетворяет своим спецификациям и готово к использованию в условиях эксплуатации. Квалификационное тестирование выполняется для каждого компонента ПО по всем разделам требований при широком варьировании тестов.

З) Установка ПО осуществляется разработчиком в соответствии с планом в той среде и на том оборудовании, которые предусмотрены договором. В процессе установки проверяется работоспособность ПО и баз данных.

И) Приемка ПО предусматривает оценку результатов квалификационного тестирования ПО и системы и документирование результатов оценки, которые проводятся заказчиком с помощью разработчика.

Создание архитектуры приложения — это процесс формирования структурированного решения, отвечающего всем техническим и операционным требованиям и обеспечивающего оптимальные общие атрибуты качества, такие как производительность, безопасность и управляемость. Он включает принятие ряда решений на основании широкого диапазона факторов. Каждое из этих решений может иметь существенное влияние на качество, производительность, удобство обслуживания и общий успех приложения.

Как и любая другая сложная структура, программное обеспечение (ПО) должно строиться на прочном фундаменте. Неправильное определение ключевых сценариев, неправильное проектирование общих вопросов или неспособность выявить долгосрочные последствия основных решений могут поставить под угрозу все приложение. Современные инструменты и платформы упрощают задачу по созданию приложений, но не устраняют необходимости в тщательном их проектировании на основании конкретных сценариев и требований. Неправильно выработанная архитектура обусловливает нестабильность ПО, невозможность поддерживать существующие или будущие бизнес-требования, сложности при развертывании или управлении в среде производственной эксплуатации. Проектирование систем должно осуществляться с учетом потребностей пользователя, системы (ИТ-инфраструктуры) и бизнес-целей. Для каждой из этих составляющих определяются ключевые сценарии и выделяются важные параметры качества (например, надежность или масштабируемость), а также основные области удовлетворенности и неудовлетворенности. По возможности необходимо выработать и учесть показатели успешности в каждой из этих областей.

Основное назначение архитектуры — описание использования или взаимодействия основных элементов и компонентов приложения. Выбор структур данных и алгоритмов их обработки или деталей реализации отдельных компонентов являются вопросами проектирования. Часто вопросы архитектуры и проектирования пересекаются. Вместо того чтобы вводить жесткие правила, разграничивающие архитектуру и проектирование, имеет смысл комбинировать эти две области.

В некоторых случаях, принимаемые решения, очевидно, являются архитектурными по своей природе, в других — больше касаются проектирования и реализации архитектуры. Приступая к работе над архитектурой приложения, необходимо помнить об основных принципах проектирования. Это поможет создать архитектуру, которая будет следовать проверенным подходам, обеспечит минимизацию затрат, простоту обслуживания, удобство использования и расширяемость.

Рассмотрим основные принципы:

1. Разделение функций. Разделите приложение на отдельные компоненты, по возможности, минимальным перекрытием функциональности. Важным фактором является предельное уменьшение количества точек соприкосновения, что обеспечит высокую связность (high cohesion) и слабую связанность (low coupling). Неверное разграничение функциональности может привести к высокой связанности и сложностям взаимодействия, даже несмотря на слабое перекрытие функциональности отдельных компонентов.

2. Принцип единственности ответственности. Каждый отдельно взятый компонент или модуль должен отвечать только за одно конкретное свойство/функцию или совокупность связанных функций.

3. Принцип минимального знания (также известный как Закон Деметера (Law of Demeter, LoD)). Компоненту или объектуне должны быть известны внутренние детали других компонентов или объектов.

4. Не повторяйтесь. Намерение должно быть обозначено только один раз. В применении к проектированию приложения это означает, что определенная функциональность должна быть реализована только в одном компоненте и не должна дублироваться ни в одном другом компоненте.

5. Минимизируйте проектирование наперед. Проектируйте только то, что необходимо. В некоторых случаях, когда стоимость разработки или издержки в случае неудачного дизайна очень высоки, может потребоваться полное предварительное проектирование и тестирование. В других случаях, особенно при гибкой разработке, можно избежать масштабного проектирования. Если требования к приложению четко не определены, или существует вероятность изменения дизайна со временем, старайтесь не тратить много сил на проектирование раньше времени.

Цель архитектора ПО при проектировании приложения или системы — максимальное упрощение дизайна через его разбиение на функциональные области. Например, пользовательский интерфейс (user interface, UI), выполнение бизнеспроцессов или доступ к данным — все это разные функциональные области. Компоненты в каждой из этих областей должны реализовывать данную конкретную функциональность и не должны смешивать в себе код разных функциональных областей. Так в компонентах UI не должно быть кода прямого доступа к источнику данных; для извлечения данных в них должны использоваться либо бизнес-компоненты, либо компоненты доступа к данным.

Также необходимо проанализировать соотношение затрат/выгод для инвестиций в приложение. В некоторых случаях может быть целесообразным упростить структуру и разрешить, например, связывание элементов UI с результирующими данными. В общем, оценивайте реализацию функциональности также и с коммерческой точки зрения. Далее приводятся обобщенные рекомендации, которые помогут учесть широкий диапазон факторов, влияющих на проектирование, реализацию, развертывание, тестирование и обслуживание приложения.

Архитектура программного обеспечения заключает в себе ряд важных решений об организации программной системы, среди которых выбор структурных элементов и их интерфейсов, составляющих и объединяющих систему в единое целое; поведение, обеспечиваемое совместной работой этих элементов; организацию этих структурных и поведенческих элементов в более крупные подсистемы, а также архитектурный стиль, которого придерживается данная организация.

Выбор архитектуры ПО также касается функциональности, удобства использования, устойчивости, производительности, повторного использования, понятности, экономических и технологических ограничений, эстетического восприятия и поиска компромиссов.

Архитектурный стиль, иногда называемый архитектурным шаблоном — это набор принципов, высокоуровневая схема, обеспечивающая абстрактную инфраструктуру для семейства систем. Архитектурный стиль улучшает секционирование и способствует повторному использованию дизайна благодаря обеспечению решений часто встречающихся проблем. Архитектурные стили и шаблоны можно рассматривать как набор принципов, формирующих приложение.

Понимание архитектурных стилей обеспечивает несколько преимуществ. Самое главное из них — общий язык. Также они дают возможность вести диалог, не касаясь технологий, т. е. обсуждать схемы и принципы, не вдаваясь в детали. Например, архитектурные стили позволяют сравнивать схему клиент/сервер с n-уровневой схемой приложения. Архитектурные стили можно организовать по их фокусу.

В таблице приведен список типовых архитектурных стилей, и дается краткое описание каждого из них.

|  |  |
| --- | --- |
| Архитектурный стиль/парадигма | Описание |
| Клиент/сервер | Система разделяется на два приложения, где клиент выполняет запросы к серверу. Во многих случаях в роли сервера выступает база данных, а логика приложения представлена процедурами хранения |
| Компонентная архитектура | Дизайн приложения разлагается на функциональные или логические компоненты с возможностью повторного использования, предоставляющие тщательно проработанные интерфейсы связи |
| Дизайн на основе предметной области | Объектно-ориентированный архитектурный стиль, ориентированный на моделирование сферы деловой активности и определяющий бизнес-объекты на основании сущностей этой сферы |
| Многослойная архитектура | Функциональные области приложения разделяются на многослойные группы (уровни) |
| Шина сообщений | Архитектурный стиль, предписывающий использование программной системы, которая может принимать и отправлять сообщения по одному или более каналам связи, так что приложения получают возможность взаимодействовать, не располагая конкретными сведениями друг о друге |
| N-уровневая / 3-уровневая | Функциональность выделяется в отдельные сегменты, во многом аналогично многослойному стилю, но в данном случае сегменты физически располагаются на разных компьютерах. |
| Объектно-ориентированная | Парадигма проектирования, основанная на распределении ответственности приложения или системы между отдельными многократно используемыми и самостоятельными объектами, содержащими данные и поведение |
| Сервисно-оринетрированная архитектура (SOA) | Описывает приложения, предоставляющие и потребляющие функциональность в виде сервисов с помощью контрактов и сообщений |

Архитектура программной системы практически никогда не ограничена лишь одним архитектурным стилем, зачастую она является сочетанием архитектурных стилей, образующих полную систему. Например, может существовать SOA-дизайн, состоящий из сервисов, при разработке которых использовалась многослойная архитектура и объектно-ориентированный архитектурный стиль.

Рассмотрим основные базовые типы приложений:

1. Мобильные приложения. Приложения этого типа могут разрабатываться как тонкий клиент или насыщенное клиентское приложение. Насыщенные клиентские мобильные приложения могут поддерживать сценарии без постоянного подключения или без подключения вообще. Веб-приложения или тонкие клиентские приложения поддерживают только сценарии с подключением. Ограничением при разработке мобильных приложений могут быть устройства, на которых их предполагается выполнять.

2. Насыщенные клиентские приложения. Приложения этого типа обычно разрабатываются как самодостаточные приложения с графическим пользовательским интерфейсом, который обеспечивает отображение данных с помощью набора элементов управления. Насыщенные клиентские приложения могут поддерживать сценарии без подключения или без постоянного подключения, если должны выполнять доступ к удаленным данным или функциональности.

3. Насыщенные Интернет-приложения. Приложения этого типа могут поддерживать множество платформ и браузеров. Насыщенные Интернет-приложения выполняются в изолированной программной среде браузера, которая ограничивает доступ к некоторым возможностям клиента.

4. Сервисные приложения. Сервисы предоставляют бизнес-функциональность для совместного использования и позволяют клиентам доступ к ней из локальной или удаленной системы. Вызов операций сервиса осуществляется с помощью сообщений, соответствующих XML-схемам и передаваемых по транспортным каналам. Целью данного типа приложений является обеспечение слабой связанности между клиентом и сервером.

5. Веб-приложения. Приложения этого типа, как правило, поддерживают сценарии с постоянным подключением и различные браузеры, выполняющиеся в разнообразнейших операционных системах и на разных платформах.

6. Существует множество других более специализированных типов приложений. Как правило, эти типы являются специализациями или сочетаниями базовых типов, перечисленных в данном списке.

В таблице перечислены преимущества и недостатки общих архетипов приложений.

|  |  |  |
| --- | --- | --- |
| Тип приложения | Преимущества | Недостатки |
| Насыщенные клиентские приложения | Возможность использования ресурсов клиента. Лучшее время отклика, насыщенная функциональность UI и улучшенное взаимодействие с пользователем. Очень динамичное взаимодействие с коротким временем отклика. Поддержка сценариев без подключения и сценариев без постоянного подключения | Сложность развертывания; при этом широкий выбор вариантов установки, таких как ClickOnce, Windows Installer и XCOPY. Сложности обеспечения совместимости версий. Зависимость от платформы |
| Мобильные приложения | Поддержка портативных устройств. Доступность и простота использования для мобильных пользователей. Поддержка сценариев без подключения и сценариев без постоянного подключения | Ограниченные возможности ввода и навигации. Ограниченная область отображения экрана |
| Насыщенные Интернетприложения (RIA) | Такие же насыщенные возможности пользовательского интерфейса, как и для насыщенных клиентов. Поддержка насыщенных и потоковых мультимедиа и графики. Простота развертывания с возможностями распределения (насыщенными) такими же, как и для Веб-клиентов. Простота обновления и смены версий. Поддержка различных платформ и браузеров | Больший объем памяти, занимаемый на клиенте, по сравнению с Веб-приложением. Ограниченное использование ресурсов клиента по сравнению с насыщенным клиентским приложением. Необходимость развертывания на клиенте подходящей среды выполнения |
| Сервисные приложения | Слабо связанное взаимодействие между клиентом и сервером. Могут использоваться различными и невзаимосвязанными приложениями. Поддержка для обеспечения возможности взаимодействия | Отсутствие поддержки UI. Зависимость от возможности сетевого подключения |
| Веб-приложения | Широко доступный и основанный на стандартах UI, поддерживаемый на многих платформах. Простота развертывания и внесения изменений | Необходимость устойчивого сетевого подключения. Сложно обеспечить насыщенный пользовательский интерфейс |

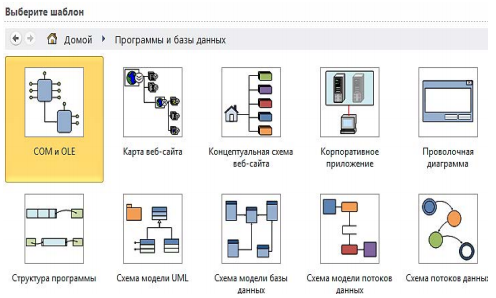
Каждый тип приложения может быть реализован с использованием одной или более технологий. Выбор технологии будет определяться сценариями и ограничениями технологий, а также возможностями и опытом группы разработки.

**Методика выполнения**

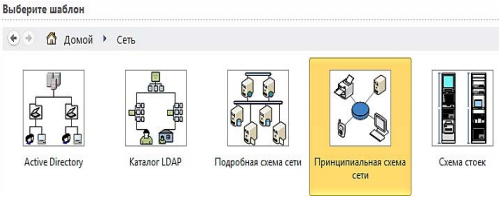
**Построение архитектуры программного средства**

Визуальное проектирование может быть осуществлено с помощью программ, поддерживающих базовые функции по построению диаграмм. Рассмотрим основные элементы, которые можно использовать для построения архитектуры в Microsoft Visio.

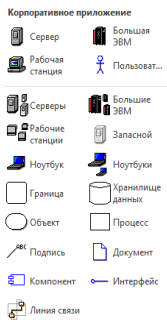
При разработке основные элементы можно взять из шаблонов «Программы и базы данных».



Дополнительно можно использовать элементы из шаблонов «Сеть».



При проектировании архитектуры используются блоки: «Процесс», «Объект», «Компонент», «Линия связи» и т. п.



Пример разработанной архитектуры приложения

