


ЖИЗНЕННЫЙ ЦИКЛ СОЗДАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



Постоев Дмитрий
postoev@okbsapr.ru

Часть 4:

Дизайн и проектирование

Определения

Архитектура программы — структура системы, которая включает в себя, элементы программы, внешние свойства элементов и отношения между ними.^[1]

Проектирование ПО — процесс определения структуры итогового решения, которое удовлетворяет всем техническим и операционным требованиям, улучшая производительность, защищенность и управляемость.^[2]

[1] Software Architecture in Practice (2nd edition), Bass, Clements, and Kazman

[2] Software Architecture and the UML, Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman

Цели проектирования

1. Выявить структуру системы, но скрыть детали реализации.
2. Выполнить требования всех заинтересованных лиц.
3. Удовлетворить функциональные и качественные требования.
4. Реализовать все сценарии использования, согласно бизнес-требований.

Движущие силы проектирования

1. Расширение возможностей пользователей (свобода действий).
2. Использование накопленного опыта (не создавать «велосипед»).
3. Гибкая структура.
4. Понимание будущих векторов развития, рыночных трендов.

Принципы проектирования

1. Разделение функциональности
2. Принцип единственной ответственности
3. «Закон Деметры» (принцип наименьших знаний)
4. DRY («Не повторяйся»)
5. YAGNI («Вам это не понадобится»)

Этапы проектирования

1. Определить тип приложения
2. Определить части системы, их взаимодействие и варианты расположения
3. Определить общие для различных частей концепции
4. Определить подходящие технологии

Типы приложений

Тип приложения	Преимущества	Ограничения
Мобильное приложение	<ol style="list-style-type: none">1. Поддержка носимых устройств2. Подходит для оффлайн и периодического онлайн доступа	<ol style="list-style-type: none">1. Ограниченная управляемость и ввод данных2. Размеры экрана
«Толстое» клиентское приложение	<ol style="list-style-type: none">1. Возможность использовать ресурсы клиента2. Богатая UI функциональность3. Быстрый отклик и лучшая доступность4. Подходит для оффлайн доступа	<ol style="list-style-type: none">1. Тяжело разворачивать2. Сложность поддержки3. Платформозависимое
«Толстое» Интернет-приложение	<ol style="list-style-type: none">1. Богатая UI функциональность2. Простота разворачивания и обновления3. Кроссплатформенность	<ol style="list-style-type: none">1. Требуется поддержки на стороне клиента (по сравнению с Web- приложением)2. Ограничения на использования клиентских ресурсов3. Необходима поддержка подходящего окружения на стороне клиента
Сервисное приложение	<ol style="list-style-type: none">1. Слабо зависимое взаимодействие клиента и сервера2. Могут быть использованы разными, несвязанными приложениями3. Поддержка совместимости	<ol style="list-style-type: none">1. Нет поддержки UI2. Зависят от сетевого соединения
Web-приложение	<ol style="list-style-type: none">1. Широкая поддержка стандартизированных UI на различных платформах2. Простота разворачивания и обновления	<ol style="list-style-type: none">1. Зависят от непрерывного сетевого соединения2. Тяжело предоставить богатую UI функциональность

Архитектурные паттерны

— высокоуровневый шаблон, улучшающий разделение и способствующий повторному использованию дизайна для повторяющихся проблем.^[1]

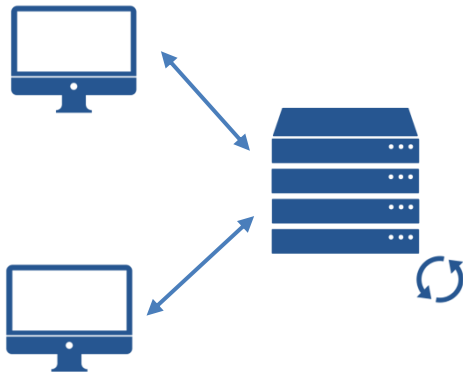
Категория	Паттерны
Сетевое взаимодействие	Сервис-ориентированная архитектура (Service-Oriented Architecture) Сервисная шина (Message Bus) Брокер сообщений (Message Broker) Издатель/Подписчик (Publish/Subscribe)
Разделение функциональности	Клиент/Сервер (Client/Server) Многослойная архитектура (N-Tier)
Домен	Проблемно-ориентированное проектирование (Domain-Driven Design)
Структура	Многослойная архитектура (Layered) Объектно-ориентированная (Object-Oriented) Компонентная (Component-Based)

[1] Microsoft Application Architecture Guide, 2nd Edition, 2009

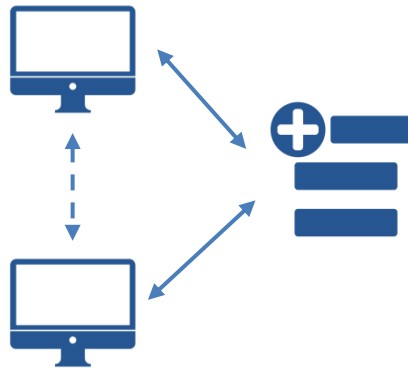
Шаблон «Клиент/Сервер»

- архитектурный паттерн, описывающий распределенную систему, включающую клиента (клиентов) и сервер, соединенные сетью.

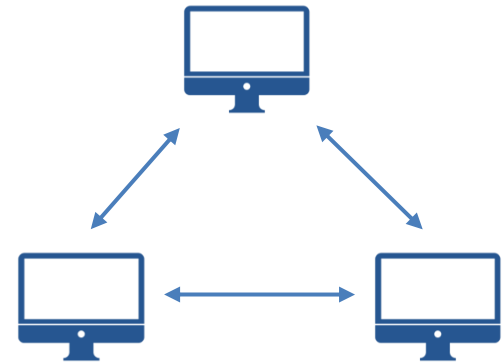
Client-Server



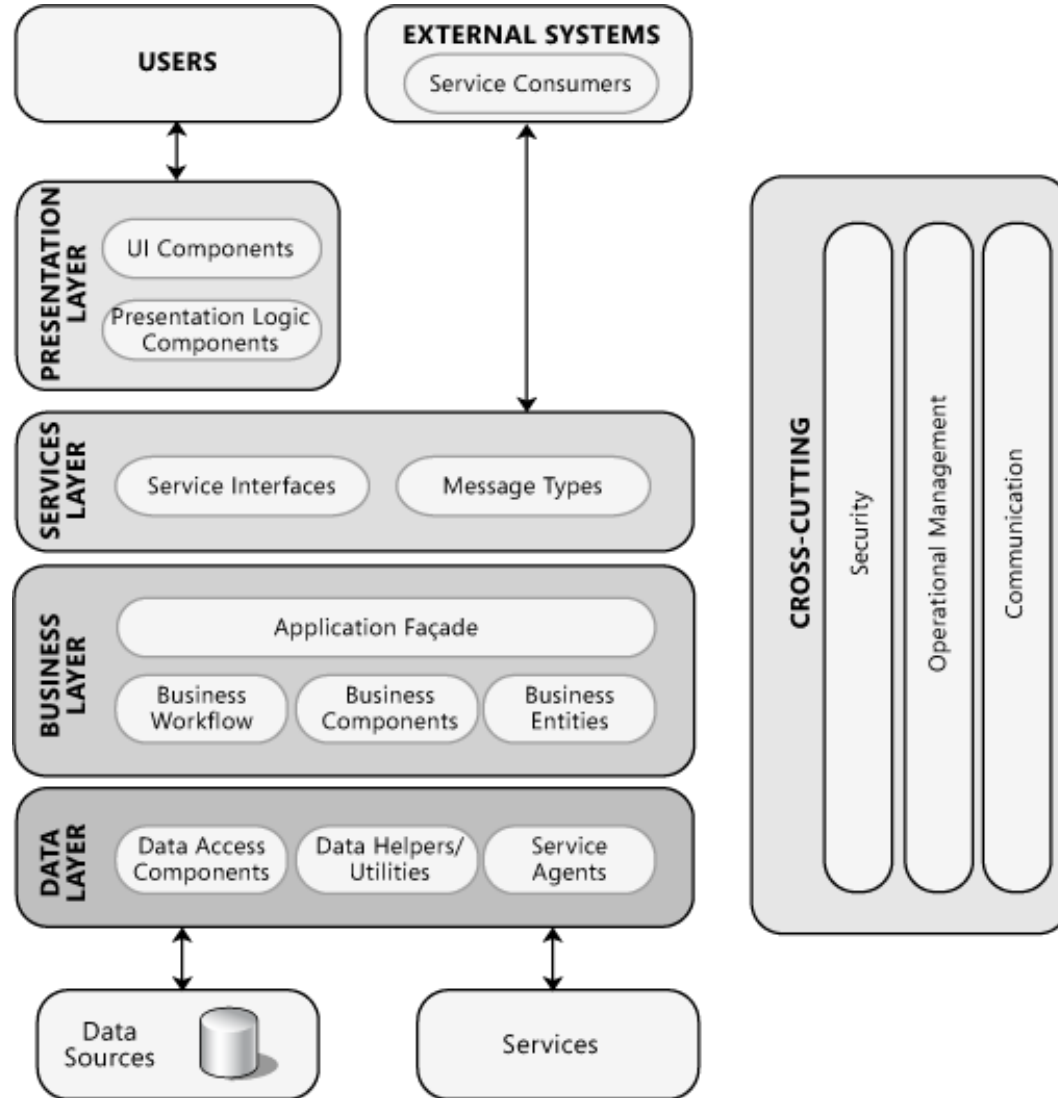
Client-Queue-Client



Peer-to-Peer

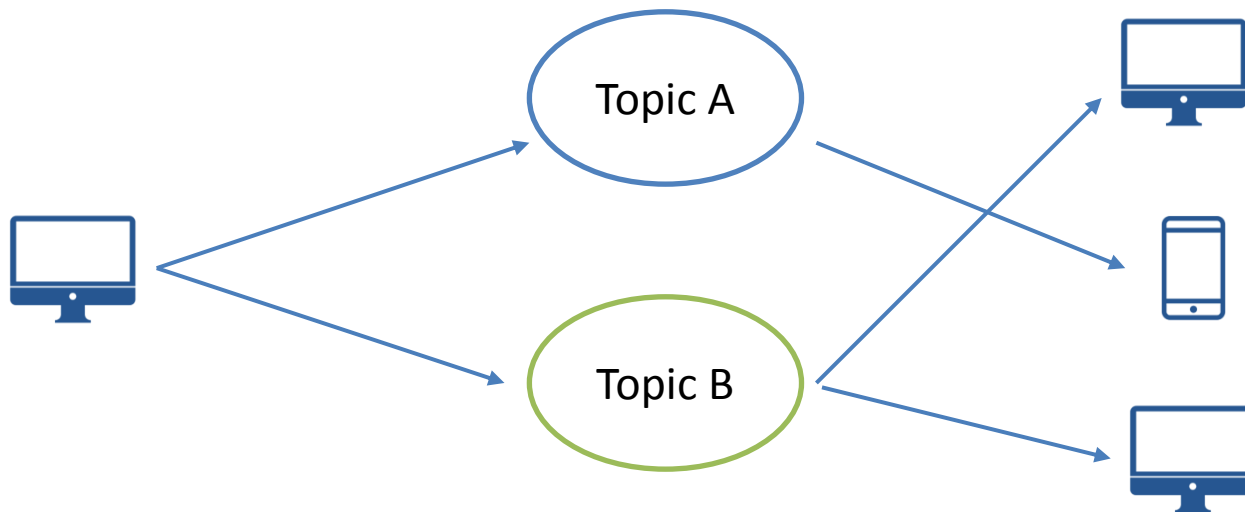


Шаблон «Многослойная Архитектура»



Шаблон «Издатель-Подписчик»

- шаблон проектирования передачи сообщений, в котором отправители сообщений, именуемые издателями, напрямую не привязаны программным кодом отправки сообщений к подписчикам.



Комплексные вопросы

1. Конфигурирование и настройка системы
2. Журналирование
3. Протокол взаимодействия и формат передаваемых данных
4. Обработка исключительных ситуаций и восстановление после сбоя
5. Проверка корректности данных
6. Безопасность
7. Контроль состояния и транзакции
8. Кэширование и оптимизация

Выбор подходящих технологий

Раздел	Библиотека
Сетевое взаимодействие	ZeroMQ (czmq), Paho MQTT client, WinSocket, nanomsg, rabbitmq-c Boost.Asio, cpp-netlib, Qt, libevent, gRPC (C++)
GUI	GTK+, IUP Qt, wxWidgets, MFC (C++)
База данных	SQLite, Postgres, Redis, libmongoc
Работа с форматами	json-c, cJSON, libxml2, pugixml
Сериализация	mpack, protobuf-c
Безопасность	OpenSSL, libgcrypt
Получение информации о системе	WinAPI (C++)
Полезное	Glib, APR, EFL

1. «Microsoft Application Architecture Guide, 2nd Edition», 2009
2. «Software Architecture in Practice, 2nd ed», Bass, Len, Paul Clements, and Rick Kazman, 2003
3. «Patterns of Enterprise Application Architecture», Fowler Martin, 2002
4. «Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process», Ambler, Scott. J. Wiley, 2002.
5. «Evaluating Software Architectures: Methods and Case Studies», Clements, Paul, Rick Kazman, and Mark Klein

Полезные ссылки

1. Написание Windows сервисов -
[https://msdn.microsoft.com/en-us/library/bb540476\(v=VS.85\).aspx](https://msdn.microsoft.com/en-us/library/bb540476(v=VS.85).aspx)
2. ZeroMQ Guide (сетевая библиотека) -
<http://zguide.zeromq.org/page:all>
3. Qt (набор библиотек для C++) - <https://www.qt.io/ru/>
4. Получение информации о процессах и их ресурсах (C++) -
[https://msdn.microsoft.com/en-us/library/ms684884\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/ms684884(VS.85).aspx)

Домашнее задание

1. **Руководитель проекта** должен назначить задачу (*Issues*) **Программистам** на разработку Архитектуры программного продукта на основе описанных ранее требований.
2. **Аналитик** дополняет спецификацию требований программного обеспечения и устраняет выявленные замечания (замечания обозначаются Руководителем проекта как *Issues*).

Примечание: Задачи назначаются и оформляются на Github