# ПРОЕКТИРОВАНИЕ МНОГОУРОВНЕВОЙ АРХИТЕКТУРЫ ПРОГРАММНОГО СРЕДСТВА ПРОВЕДЕНИЯ ЭЛЕКТРОННЫХ АУКЦИОНОВ В РЕЖИМЕ ОНЛАЙН

*Белорусский государственный университет информатики и радиоэлектроники*
*г. Минск, Республика Беларусь*

*Сергушов Н. О.*

В статье рассматриваются основные принципы построения многоуровневой архитектуры программного средства.

Многоуровневая архитектура – одна из архитектурных парадигм разработки. Хорошая архитектура это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать. То есть, на самом деле можно сформулировать список вполне разумных и универсальных критериев:

• Эффективность системы. В первую очередь программа, конечно же, должна решать поставленные задачи и хорошо выполнять свои функции, причем в различных условиях. Сюда можно отнести такие характеристики, как надежность, безопасность, производительность, способность справляться с увеличением нагрузки (масштабируемость) и т.п.

• Гибкость системы. Любое приложение приходится менять со временем — изменяются требования, добавляются новые. Чем быстрее и удобнее можно внести изменения в существующий функционал, чем меньше проблем и ошибок это вызовет — тем гибче и конкурентоспособнее система. Поэтому в процессе разработки старайтесь оценивать то, что получается, на предмет того, как вам это потом, возможно, придется менять. Спросите у себя: «А что будет, если текущее архитектурное решение окажется неверным?», «Какое количество кода подвергнется при этом изменениям?». Изменение одного фрагмента системы не должно влиять на ее другие фрагменты. По возможности, архитектурные решения не должны «вырубаться в камне», и последствия архитектурных ошибок должны быть в разумной степени ограничены.

• Расширяемость системы. Возможность добавлять в систему новые сущности и функции, не нарушая ее основной структуры. На начальном этапе в систему имеет смысл закладывать лишь основной и самый необходимый функционал (принцип YAGNI — you ain’t gonna need it, «Вам это не понадобится») Но при этом архитектура должна позволять легко наращивать дополнительный функционал по мере необходимости. Причем так, чтобы внесение наиболее вероятных изменений требовало наименьших усилии.

• Тестируемость. Код, который легче тестировать, будет содержать меньше ошибок и надежнее работать. Но тесты не только улучшают качество кода. Многие разработчики приходят к выводу, что требование «хорошей тестируемости» является также направляющей силой, автоматически ведущей к хорошему дизайну, и одновременно одним из важнейших критериев, позволяющих оценить его качество: "Используйте принцип «тестируемости» класса в качестве «лакмусовой бумажки» хорошего дизайна класса. Даже если вы не напишите ни строчки тестового кода, ответ на этот вопрос в 90% случаев поможет понять, насколько все «хорошо» или «плохо» с его дизайном".

Систему желательно проектировать так, чтобы ее фрагменты можно было повторно использовать в других системах. Над программой, как правило, работает множество людей — одни уходят, приходят новые. После написания сопровождать программу тоже, как правило, приходится людям, не участвовавшем в ее разработке. Поэтому хорошая архитектура должна давать возможность относительно легко и быстро разобраться в системе новым людям. Проект должен быть хорошо структурирован, не содержать дублирования, иметь хорошо оформленный код и желательно документацию. И по возможности в системе лучше применять стандартные, общепринятые решения привычные для программистов. Чем экзотичнее система, тем сложнее ее понять другим (Принцип наименьшего удивления — Principle of least astonishment. Обычно, он используется в отношении пользовательского интерфейса, но применим и к написанию кода).

Многоуровневая архитектура – одна из архитектурных парадигм разработки ПО, при которой разбиение приложения на самостоятельные составные части происходит по реализуемой ими функциональности.

Характерные особенности многоуровневой архитектуры:

• Требуемая функциональность реализуется в одном уровне и не дублируется в других;

• Разделение возможностей приложения ПО функциональным областям: пользовательский интерфейс, сервисный слой, бизнес-логика, доступ к данным и др;

• Каждый уровень должен чётко реализовывать ту функциональность, к области которой он относится, не совмещая код других функциональных областей

• Организация передачи данных между слоями;

• Уровни слабо связаны, данными обмениваются явно;

• Каждый слой агрегирует ответственности и абстракции уровня, расположенного непосредственно под ним;

• Физически все слои могут быть развёрнуты на одном компьютере или распределены по разным компьютерам;

• Использование в других системах;

• Уровень доступа к данным содержит методы, обрабатывающие вставку, выборку, изменение или удаление данных в БД, известных как CRUD методы;

• Уровень бизнес-логики реализует функциональные возможности приложения;

• Уровень пользовательского интерфейса предоставляет эргономичный интерфейс пользователю в соответствии с функционалом, описанном в техническом задании;

К главным качествам многоуровневой архитектуры можно отнести:

• Масштабируемость (Scalability) ¬ озможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей.

• Ремонтопригодность (Maintainability) ¬ изменение одного модуля не требует изменения других модулей

• Заменимость модулей (Swappability) ¬ модуль легко заменить на другой

• Возможность тестирования (Unit Testing) ¬ модуль можно отсоединить от всех остальных и протестировать / починить

• Переиспользование (Reusability) ¬ модуль может быть переиспользован в других программах и другом окружении

• Сопровождаемость (Maintenance) ¬ разбитую на модули программу легче понимать и сопровождать

Таким образом, хорошая архитектура это, прежде всего, модульная/блочная архитектура. Чтобы получить хорошую архитектуру надо знать, как правильно делать декомпозицию системы. А значит, необходимо понимать — какая декомпозиция считается «правильной» и каким образом ее лучше проводить.

Список используемых источников:

1. Марко Беллиньясо. Разработка Web-приложений в среде ASP.NET 2.0 – 2007 г.
2. Мартин Фаулер. Архитектура корпоративных программных приложений – 2007 г.