Архитектура программного обеспечения.

Конспект на основе различных источников.

1. Архитектура программного обеспечения. Определение и использование.

Системой называется интегрированный комплекс, состоящий из одного или более процессов, аппаратных устройств, программ, средств и людей, предоставляющий возможность удовлетворить данную потребность или условие, стандарт IEEE 12207.

Архитектура - базовая реализация системы, воплощенная в ее компонентах, в их отношениях между собой и с окружением, а также определяющие проектирование и развитие системы.

Термин архитектура в программном обеспечении подразумевает набор решений по поводу организации системы, набор структурных элементов компоновки системы и их интерфейсов, при помощи которых система собирается воедино; поведение системы определяется взаимодействиями компонентов.

Архитектура обладает свойствами укрупнения и разделения на уровни масштаба, для контроля над организацией такой системы используются архитектурные стили. Выбор архитектурного стиля производят заинтересованные в ней лица на основании логического обоснования.

Архитектура подразумевает использование термина структура, говоря об отдельных компонентах архитектурного решения. Структуры позволяют строить абстрактные архитектуры не вдаваясь в детали ее реализации. Структурный элемент может быть подсистемой, процессом, библиотекой, базой данных, вычислительным узлом, системой в традиционном смысле, готовым продуктом и так далее.

Некоторые определения архитектуры используют термин структурные элементы, для обозначения группы структур осуществляющих одну задачу в глобальном смысле образуя отдельное звено системы. Такое звено может представлять собой, к примеру, сокет, быть синхронным или асинхронным, быть связанным с конкретным протоколом и так далее. Еще пример структурного элемента: объединение структур заказа, управления аккаунтом и информации о потребителе.

К задачам архитектуры не всегда сводится полное определение и поведение структур. Архитектура не занимается проектированием не значимых структур и элементов.

Значимые элементы - это элементы, которые имеют продолжительное и устойчивое действие, например, главные структурные элементы, элементы, связанные с основным поведением и элементы, которые определяют значимые свойства, такие как надежность и масштабируемость. Архитектура не имеет отношения к единичным, обособленным свойствам структур. К понятию значимости в архитектурных решениях можно также отнести понятие экономической значимости, определение стоимости создания и стоимости изменения.

В итоге архитектура концентрируется на значимых элементах, перспектива оценки которой важна разработчику управлять системой и ее сложностью.

Стоит отметить что набор значимых элементов не закрепляется на этапе проектирования архитектуры и может быть изменен с ростом системы. Но чем стабильнее будет список значимых элементов, тем надежнее и более предсказуемой система вырастает в итоге.

Иногда при построении архитектуры нет возможности выполнить все пожелания заинтересованного лица. Например, соответствие определенной функции заданному промежутку времени. Возникают также ситуации, когда интересы являются нефункциональными по характеру, так как они не влияют на функциональность системы. В таком случае к задачам архитектора следует отнести поиск компромиссного решения.

Важный аспект архитектуры - это не только конечный результат, то есть сама архитектура, но и ее логическое обоснование. Таким образом, важно обеспечить документирование решений, которые привели к созданию этой архитектуры, и ло-

гические обоснования таких решений. Поскольку в этих обоснованиях будут нуждаться специалисты обслуживающие систему.

Архитектурный стиль - это общий вид шаблона используемый для решения сходного набора интересов. Хотя шаблон и является составным и способен подстраиваться под цели конкретной задачи, к его особенностям можно отнести единый предсказуемый стиль поведения. Шаблон - это общее решение общей проблемы в данном контексте.

Примеры архитектурных стилей включают распределенный стиль, стиль "каналы и фильтры", стиль с централизованной обработкой данных, стиль, построенный на правилах и так далее. Конкретная система может демонстрировать более одного архитектурного стиля.

Кроме повторного использования опыта, применение архитектурного стиля (или шаблона) несколько облегчает жизнь разработчиков, поскольку стиль обычно документирован в терминах рационального обоснования его использования (следовательно, меньше придется обдумывать) и в терминах его структуры и поведения (следовательно, придется выработать меньше документации по архитектуре, поскольку вместо этого мы может просто обратиться к стилю).

Поскольку система строится в окружении, то окружение способно влиять при принятии решения к выбору архитектуры. Такая архитектура существует в контексте окружения. Определенные факторы окружения включаются в требования бизнеса, который будет поддерживать архитектуру. К таким требованиям можно отнести, например, стандарты организации, соответствие внешним регулятивным нормам.

Существуют требования к аппаратному обеспечению, чтобы программа работала и могла приносить пользу. Следует учитывать совокупность оборудования и программных решений используемых в построении архитектуры конечного продукта и способных повлиять на ее изменение. Программное обеспечение не способно быть надежным и безопасным без опоры на аппаратные характеристики устройства.

При проектировании решения упор также делается на людей. Все клиентские интерфейсы ориентированы на конечного пользователя системы. Как правило выбор архитектуры в таком случае насчитывается на подготовленность пользователя системы, если пользователей системы много расчет идет на случайную выборку, из которой составляется портрет среднего пользователя.

Архитектура оказывает влияние на структуру коллектива разработчиков и администраторов системы. Структура групп разработчиков выравнивается в соответствии с обладаемыми навыками и способностью следовать плану архитектуры. Однако чаще встречается ситуация, когда первоначальная группа разработчиков оказывает влияние на архитектуру, а не наоборот. Это ошибка, которой следует избегать, поскольку в результате обычно получается архитектура, далекая от идеала. Но наследует думать, что всегда возможно построение идеальной архитектуры, поскольку реальная структура коллектива и доступные навыки представляют весьма ощутимое ограничение того, что могло бы быть, и разработчику следует это учитывать.

Следует также отметить, что каждая система имеет архитектуру, даже если эта архитектура формально не документирована или система слишком проста, и, скажем, состоит из одного элемента. Обычно документирование архитектуры представляет собой очень ценное средство. Документированные архитектуры имеют тенденцию быть более продуманными - а, следовательно, более эффективными - чем недокументированные, поскольку процесс записи архитектуры естественным образом ведет к всестороннему обдумыванию. И наоборот если не документировать архитектуру трудно доказать ее эффективность.

Архитектура ассоциируется со строительством инженерных сооружений. Справедливо считать что построение архитектуры по является частью инженерной работы. При разработке программного обеспечения можно выделить такие понятия, как корпоративная архитектура, системная архитектура, организационная архитектура, архитектура информации, архитектура аппаратного обеспечения, архитектура приложения, архитектура инфраструктуры и так далее.

В отрасли нет единого соглашение на значения используемых терминов, зато они обладают вполне конкретными целями и являются частью конструкторской работы.

2. Характеристики архитектуры программного обеспечения.

Архитектура программного обеспечения демонстрирует следующие характеристики: множество заинтересованных сторон, разделение проблем, ориентация на качество, повторяющиеся стили, концептуальная целостность, когнитивные ограничения.

Решение проблем лежит в основе построения архитектуры и определяет документированным способом целесообразность ее применения. Ориентация на качество подразумевает использование классических методов построения программных решений, основывались на требуемой функциональности и потоке данных через систему, текущее понимание термина состоит в наличии у разработанной системы определённых характеристик: отказоустойчивость , обратная совместимость , расширяемость , надежность , ремонтопригодность , доступность , безопасность, удобство использования и другие подобные возможности. Повторяющиеся стили подразумевают использование единых правил в архитектурном решении - единые шаблоны, повторяющиеся атрибуты, тактика, приверженность правилам и так далее. Концептуальная целостность представляет собой общее видение того, что она должна делать и как она должна это делать, основываясь на опыте архитектора, который хранит все сведения о системе. Когнитивные ограничения описывают привычки и стандарты пользователей или организаций, внедряющих программное решение.

Для построения архитектуры используются различные абстракции такие как языки описания архитектуры.

Язык описания архитектуры (ADL) - это любое средство выражения, используемое для описания архитектуры программного обеспечения (ISO / IEC / IEEE 42010). Многие специальные ADL были разработаны с 1990-х годов, в том числе AADL (стандарт SAE), Wright (разработан Карнеги-Меллоном), Acme (разработан Карнеги-Меллоном), xADL (разработан UCI), Darwin (разработан Имперским колледжем Лондона) , DAOP-ADL (разработано Университетом Малаги), SBC-ADL (разработано Национальным университетом Сунь Ят-Сена) и ByADL (Университет Л'Акуилы, Италия).

Эволюция архитектуры - это процесс поддержки и адаптации существующей архитектуры программного обеспечения для соответствия изменениям требований и среды. Поскольку программная архитектура обеспечивает фундаментальную структуру программной системы, ее развитие и обслуживание обязательно повлияют на ее фундаментальную структуру. Таким образом, эволюция архитектуры связана с добавлением новых функций, а также поддержанием существующих функций и поведения системы.

3. Подходы к созданию хорошей архитектуры программного обеспечения. Критерии хорошей архитектуры - выгодность, простота, эффективность.

Эффективность системы определяется способностью программы решать поставленные задачи согласно ожиданию пользователя. Сюда можно отнести другие характеристики по типу безопасность, производительность.

Гибкость системы определяет скорость изменения системы или ее существующего функционала. Чем быстрее это происходит - тем гибче и конкурентоспособнее система. Хорошая архитектура позволяет откладывать принятие ключевых решений и минимизирует цену ошибки.

Требование к расширяемости и гибкости системы является важным для построения архитектуры, в результате были сформулированы определенные принципы соответствия этим требованиям.

Приложение нужно проектировать так, чтобы изменения его поведения и добавление новой функциональности достигалось бы за счет написания нового кода (расширения), и при этом не приходилось бы менять уже существующий код. Важно чтобы не было модификации существующей логики при добавлении новой.

Масштабируемость процесса разработки является важным критерием хорошей архитектуры, следует обращать внимание на то, сможет ли новый человек на проекте ускорить разработку сервиса.

Если архитектура хорошо разделяема на модули, тогда можно легко тестировать эти модули в своей замкнутой системе. Тесты способны предотвратить трату времени на этапе публикации программного обеспечения, если ПО не соответствует стандарту надежности и предсказуемости это сразу будет выявлено.

Возможность повторного использования. Отличает хорошую архитектуру системы от плохой. Часто возникает задача написания однообразных элементов, задача архитектуры обеспечить отдалённость и переиспользуемость таких элементов.

Архитектура иногда нуждается в сопровождении, люди сопровождающие систему должны иметь возможность быстро разобраться в системе. Ключевой критерий в таком случае - это принцип наименьшего удивления. Задача которого придерживаться общепринятого стандарта архитектуры.

4. Критерии создания плохого дизайна архитектуры.

Архитектура тяжело изменяема. Система как цепочка, один элемент зависит от других элементов, изменение в одном из них слишком сильно влияет на поведение другой системы.

Хрупкость системы тоже плохой пример проектирования архитектуры. Система не должна ломаться от минимальных изменений.

Код программы тяжело переиспользовать, так как решения являются частными.

5. Основы декомпозиции архитектуры

Основная цель декомпозиции архитектуры - это снижение сложности при разработке большой системы. Самый лучший способ избавляйся от сложности это дробление на компоненты, то есть использование и иерархической декомпозиции. Декомпозиция должна использоваться как обязательная часть построения архитектуры. Чем более замкнутые и обособленные участки удается выделить, тем проще сделать их надёжными и безопасными.

Декомпозиция превращает код в конструктор, то есть создают уровень абстракции для использования и контроля его архитектором системы.

6. Правильная декомпозиция

Разделение программы на классы и восприятие этих классов, как отдельных модулей системы не является правильной декомпозицией. Прежде всего декомпозиция подразумевает выделение по функциональным модулям. Построение архитектуры начинается с верху вниз, а не наоборот. То есть план архитектора состоит из пунктов выделения системы, разделения ее на подсистемы или пакеты, разделение пакетов на классы.

Критерий оценки декомпозиции - фокус модулей на одной своей задаче. В определении это критерия помогают такие параметры как целостность и завер-

шенность, наличие одного входа и одного выхода, логическая независимость, слабые информационные связи с другими модулями.

Как долбится слабой связанности модулей? Во-первых, использование абстрактных классов и интересов, модули должны быть друг для друга черными ящиками, работа должна происходить с абстракциями. И каждый модуль должен иметь такой интерфейс полно описанным. Интерфейсы способ соблюдения принципа открытости закрытости системы. Они позволяют оперировать абстракциями. При ведении таких абстракций нужно соблюдать также принцип разделения интересов, чтобы не компоновать все возможности системы воедино и иметь возможность переиспользвания.

7. Использование фасадов

Фасады это интерфейсы или группы интерфейсов скрывающие за собой реализацию подсистемы. Задача фасадов - служить точкой входа в подсистему, обеспечивать ее защиту от других компонентов. Фасад хороший патерн используемый архитекторами.

Формально, требование, чтобы модули не содержали ссылок на конкретные реализации, а все зависимости и взаимодействие между ними строились исключительно на основе абстракций, то есть интерфейсов, выражается принципом инвестирование зависимостей — последний из пяти принципов SOLID). У этого принципа не самая очевидная формулировка, но суть его, как и было сказано, выражается правилом: «Все зависимости должны быть в виде интерфейсов».

Если построение независимой архитектуры подразумевает соблюдение использования абстракций объектов, то где применять сами объекты? Сами объекты должны использоваться в реализации сервисов. Модуль сам создает объекты необходимые ему для работы, но модуль не должен делать это на прямую, он должен использовать фабрики создания объектов. Модуль также может заимствовать объекты другого модуля которые уже существуют. Для обмена такими объектами применяются общие хранилища. Такое хранилище носит название локатор сервисов. Отличие локатора от фабрики это наделение объектов разными правами собственности. В то же время не стоит использовать локатор сервисов если систему можно построить без него.

Задача модуля не заботится о зависимостях, а получать их из других модулей в процессе работы. Это самое гибкое решение, дающее модулям наибольшую автономность. Можно сказать что в таком случае модуль в полной мере соблюдает принцип единой ответственности.

8. Замена прямых зависимостей обменом сообщений

Модель передачи сообщений существует для оповещения других модулей без прямых зависимостей о происходящих событиях, интересующих эти модули.

Для построения механизма обмена сообщениями используются такие шаблоны как наблюдатель или посредник.

Задача наблюдателя принимать входящие события и оповещать всех своих подписчиков об этом событии. Модель подписки подразумевает вызов определенных действий у самого подписчика, реализуя реакцию на событие.

Задача посредника стать системой обмена информацией между двумя и более модулями. Посредник становится общим ядром для синхронизации работы модулей.

9. Закон Деметры

Закон Деметры запрещает использование неявных зависимостей. Он реализует уже упоминавшийся при разборе интересов «*принцип минимального знания*», являющейся основой слабой связанности и заключающийся в том, что объект/модуль должен знать как можно меньше деталей о структуре и свойствах других объектов/модулей и вообще чего угодно, **включая собственные подкомпоненты**. Аналогия

из жизни: Если Вы хотите, чтобы собака побежала, глупо командовать ее лапами, лучше отдать команду собаке, а она уже разберётся со своими лапами сама.

10. Наследование плохая практика при построении композиции

Одну из самых сильных связей между объектами дает наследование, поэтому, по возможности, его следует избегать и заменять композицией. Для избежания ненужного наследования следует применять шаблон делегата и отдавать выполнение функций системы ему.

11. Вывод

Использование всех описанных подходов помогает архитекторам строить архитектуры систем соответсвующие стандартам индустрии.

Список источников

1. Мартин Фаулер. Архитектура корпоративных программных приложений.
2. Стив Макконнелл. Совершенный код.
3. Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования.
4. Что такое архитектура программного обеспечения? Статья: http://www.interface.ru/home.asp?artId=2116
5. Web-сайт Института разработки программного обеспечения (SEI), посвященный архитектуре - определения архитектуры http://www.sei.cmu.edu/architecture/definitions.html
6. Рекомендуемые IEEE методы описания архитектуры преимущественно-программных систем: стандарт IEEE 1472000. 2000.
7. Лен Басс, Пол Клементс, Рик Кацман , Практическая архитектура программного обеспечения, второе издание.
8. Мэри Шоу, Девид Гарлан, Архитектура программного обеспечения - перспективы рождения предмета.