17. Софтуерна архитектура. Проектиране и документиране на софтуерни архитектури.

17. Софтуерна архитектура. Проектиране и документиране на софтуерни архитектури.

С навлизането на софтуерните системи във все повече области през последните десетилетия – медицина, образование, държавна администрация, бизнес, софтуерните системи стават все по-сложни и разнообразни. Повишената комплексност на системите неминуемо води до нуждата от по-абстрактен поглед върху структурата на една система с цел тя да бъде **разбираема** от всички участници в процеса на изграждане на софтуерния продукт. Описанието на въпросния абстрактен поглед върху софтуерната система се нарича още **софтуерна архитектура**.

1. Дефиниция на софтуерна архитектура. Структури и изгледи (*structures and views*) на архитектурата.

1.1. Какво е софтуерна архитектура?

- **Дефиниция на SEI:**

"Архитектура на дадена софтуерна система е някаква съвкупност от структури, показващи различните софтуерни елементи на системата, външно видимите им свойства и връзките между тях".
"Външно видими" свойства са онези предположения, които другите елементи могат да направят относно елемента – какви услуги предлага, с какво бързодействие, как се оправя с грешките, със споделените ресурси и т.н.

- Следователно, първо и най-важно, СА е абстракция, която **скрива детайлите**, от които взаимодействието между елементите не зависи;
- Съгласно дефиницията става ясно, че системите могат да имат (и имат) **повече от една структура**. Нито една от тях самостоятелно не представлява Архитектурата на системата (структура на модулите, на процесите; елементите може да са обекти, модули, процеси, БД, библиотеки, продукти, екипи и т.н.);
- Обикновено СА се създава като **първа стъпка по време на проектирането**, като целта е да се гарантира наличието на дадени качества в системата;
- Детайли като алгоритми, представяне на данни, реализация, и т.н. не са предмет на СА;
- Предмет на СА е поведението и връзките между различни елементи, разглеждани като "черни кутии" (**systems theory**);

1.2. Какво определя софтуерната архитектура?

Изборът на софтуерна архитектура зависи от различни характеристики на средата. От друга страна избраната вече софтуерна архитектура налага известни ограничения върху средата. Тоест процесът по определяне на архитектурата и средата е цикличен (Architecture Business Cycle).

1.2.1.1. Функционалните и технически изисквания
1.2.1.2. Изменяемост и устойчивост на бизнес средата
1.2.1.3. Изменяемост, устойчивост и структура на организационната среда
1.2.1.4. Опит (с предишни системи и архитектури), знания и умения на архитекта и програмистите
1.2.1.5. Влияние на заинтересованите лица (шефове, клиенти, потребители)
1.2.1.6. Ограничения и възможности на съвременните технологии

17. Софтуерна архитектура. Проектиране и документиране на софтуерни архитектури.

1.3. От какви качества се нуждае софтуерният архитект?

В крайна сметка софтуерният архитект трябва **да балансира** избора си между очакванията на всички заинтересовани лица.
Отговорност на софтуерния архитект е **да притисне** ЗЛ да сближат позициите и очакванията си по отношение на софтуерната система.

1.3.1.1. Отлично познаване на технологиите;
1.3.1.2. Отлично аналитично мислене;
1.3.1.3. Комуникативност, дипломатичност и умения за убеждаване и водене на преговори;

1.4. Отговорности на софтуерния архитект

1.4.1.1. Вземане на бизнес решенията за създаване на системите;
1.4.1.2. Разбиране на изискванията (участие в изготвянето им);
1.4.1.3. Създаване или избор на архитектура;
1.4.1.4. Документиране и преразказване на СА (по различен начин за различните групи ЗЛ);
1.4.1.5. Анализ и оценка (архитектурна и финансова) на СА;
1.4.1.6. Създаване на системата;
1.4.1.7. Следене за наличие на съответствие между системата и СА.

1.5. Първични критерии за качество на софтуерна архитектура

1.5.1.1. Малък екип от архитекти с утвърден лидер, който взима решенията;
1.5.1.2. Добре дефинирани и приоритизирани изисквания;
1.5.1.3. СА трябва да е добре документирана и разбираема от всички ЗЛ;
1.5.1.4. ЗЛ трябва да са активно въвлечени в рецензията и одобрението на СА;
1.5.1.5. Количествени измервания за съответствие с качествените изисквания;
1.5.1.6. СА трябва да позволява постепенна реализация (обособен скелет);
1.5.1.7. Присъствие на софтуерни техники за разумно разходване на ресурси;
1.5.1.8. Модулност на СА;
1.5.1.9. Модулите, които произвеждат данни, трябва да са отделени от модулите, които консумират данни;

1.6. Структури и изгледи на архитектура

1.6.1.1. Както в медицината например, различните специалисти имат различен поглед върху човешкото тяло и всички те заедно дефинират архитектурата на Човека, така и СА предлага *няколко различни погледа (структури)* на софтуерната система;
1.6.1.2. **Структура** – съвкупност от софтуерни елементи, техните външно видими свойства и връзките между тях;
1.6.1.3. **Изглед** – конкретно документирано представяне на дадена структура;
1.6.1.4. **Структура/ Изглед** - взаимозаменяеми

1.7. Типове структури

1.7.1. **Модулни структури** (*кой модул каква функционалност реализира, кой модул с кой друг модул взаимодейства и т.н.)*

- **Декомпозиция на модулите** – разбираемост, разпределение на задачите, лесна променяемост и разширяемост на архитектурата;
- **Структура на слоевете** - Частен случай на структура на модулите е структура на слоевете, при които слой N може да ползва единствено възможностите на слой N-1;
- **Структура на употребата на модули**

17. Софтуерна архитектура. Проектиране и документиране на софтуерни архитектури.

Разработчиците знаят с кои интерфейси трябва да се запознаят, за да си свършат работата по създаването на даден модул;
Тестерите знаят къде трябва да гледат, за да тестват даден модул;

1.7.2. **Структура на компонентите и конекторите** (*процеси,изпълнявани в компонентите* и *комуникационните канали между процесите*)

- Кои са основните изчислителни процеси в системата? Как тези процеси въздействат върху данните? Кои са споделените ресурси?
- **Структура на процесите** - има отношение към *бързодействието* и *надеждността*. Избягване на deadlock.
- **Структура на конкурентното изпълнение**
- **Структура на споделените данни** – има отношение към бързодействието

1.7.3. **Структура на разположението**
(физическо, ресурсно(CPU), организационно)

- Структура на внедряването
- Разпределение на работата (модул <-> екип/способности)

2. Изисквания към качеството (нефункционални изисквания) на системата.

Качеството е субективно възприятие – различните ЗЛ могат да не одобрят даден дизайн, тъй като тяхната идея за качество се различава от идеята за качество на архитекта;

- Бизнес целите определят **Качествата**, които трябва да бъдат вградени в архитектурата на системата. Тези Качества поставят изисквания отвъд функционалните (описание на основните възможности на системата и услугите които тя предоставя);
- Въпреки, че функционалността и Качествата са тясно свързани, функционалността често е **единственото, което се взема под внимание** по време на проектирането.
- Като следствие много **системи се преправят** не защото им липсва функционалност, а защото е трудно да се поддържат, трудно е да се смени платформата, не са скалируеми, прекалено са бавни, или пък са несигурни.
- **СА е тази стъпка** в процеса на създаването на системата, в която за пръв път се разглеждат качествените изисквания и в зависимост от тях се създават съответните структури, на които се вменява функционалност.
- За да притежава дадена система изискваните качествени характеристики, те трябва да се имат предвид както по време на проектирането, така и по време на разработката и внедряването.

2.1. Сценарий за качество

- Изискванията за качество трябва да се формализират от архитекта посредством т.н. "*сценарии за качество*", за да бъдат те поставени на обективна основа;
- *Сценарият за качество е специфично изискване към поведението на системата в дадена ситуация, в светлината на дадено качество;*
- Сценариите демонстрират какво е качество в рамките на създаваната система, като дават на архитекта и на ЗЛ еднозначна основа за оценка на дизайна;

2.1.1. Характеристики

- **Въздействие** – състояние/събитие, което подлежи на обработка;
- **Източник** – обект (човек, система или нещо друго) който генерира въздействието;

- **Обект** – системата, или конкретна нейна част, върху която се случва въздействието;
- **Контекст** – Условията, при които се намира обекта по време на обработка на въздействието;
- **Резултат** – действията, предприети от обекта при случването на въздействието;
- **Количествени параметри** – резултатът трябва да подлежи на някакви количествени измервания, така че да позволи проверката дали сценарият се изпълнява съгласно изискванията;

- **Сценарий за Изменяемост (Пример)**
  - *"Преди пускането на системата в експлоатация… (Контекст)*
  - *…клиентът… (Източник)*
  - *…желае промяна на фоновия цвят на екрана….(Въздействие)*
  - *За целта се променя изходния код…. (Обект)*
  - *Това не трябва да предизвиква никакви странични ефекти в поведението на системата. (Резултат)*
  - *Времето за извършване на промяната, вкл. тестването й трябва да отнеме по малко от 3 часа*
  - *(Количествени параметри)*

2.2. Взаимовръзка между качествата

Трябва да се има предвид, че качествата са взаимносвързани и завишаването на едно качество, води до понижаване на друго.

- **Сигурност vs. Отказоустойчивост (Fault Tolerance)** – въпреки, че обикновено ги поставят в една и съща графа, те си противоречат – сигурността изисква наличието на single point of failure, т.е. системата да може да се компроментира от едно единствено място; Обратно, отказоустойчивостта предполага репликация
- **Преносимост vs. Производителност** – основната техника за постигането на преносимост е изолирането на зависимостта от ОС и хардуер в специализирани модули, което внася допълнителна тежест по време на изпълнението и намалява производителността;

2.3. Технологични качества

2.3.1. Надеждност (Reliability)

2.3.2. Изменяемост (Modifiability)

Определя се най-вече от декомпозицията. Свързва с цената на промените.

2.3.3. Производителност (Performance)

Времето, за което системата реагира на дадени събития. Зависи от комуникацията между компонентите, консумацията на споделени ресурси, разпределението на функционалността между компонентите

2.3.4. Сигурност (Security)

Способността на системата да устоява на опити за неразрешена употреба.

2.3.5. Изпитаемост (Testability)

Лекотата, с която системата може да бъде изтествана.

2.3.6. Готовност (Availability)
Честота на сривове и време за оправяне на повредата.

2.3.7. Използваемост (Usability)

2.4. Бизнес качества

2.4.1. Време за пускане на пазара (Time to market TTM)
Намалява се с употребата на COTS продукти.

2.4.2. Себестойност и печалба
Зависи от архитектурата => технология (непозната, изменяема)

2.4.3. Предвидено време за живот (Time to live TTL)

2.4.4. Възможност за постепенно пускане на пазара (extensibility)

2.5. Архитектурни качества

2.5.1. Идейна цялост
Всички да са наясно с общата идея/визия на СА. Подобни задачи да се решават с подобни решения.

2.5.2. Коректност и пълнота спрямо изискванията

2.5.3. Градивност (с каква лекота и в какъв срок наличният екип ще построи системата)

3. Проектиране на софтуерната архитектура. Процес за проектиране. Избор на подходящи структури. Последователност на създаване на архитектурата. Тактики (архитектурни решения) за постигане на желаните качествени показатели.
При наличието на основните функционални и технически изисквания може да се стартира процеса по проектиране на СА. В началото се избират до 10-тина основополагащи функционални, технически и бизнес изисквания, наричани **архитектурни драйвери**. Въпросните изисквания са с най-високи приоритет при проектирането на СА. Анализ на въпросните драйвери и решение на евентуални конфликти между тях. В следствие на анализа някои изисквания може да се променят.

3.1. Attribute-Driven Development (ADD)
- Подход за проектиране, в който главна роля играят качествените свойства (атрибути);
- Рекурсивен процес на дефиниране на архитектурата, като на всяка стъпка се избират конкретни техники и архитектурни модели за постигане на желаните качествени свойства;

17. Софтуерна архитектура. Проектиране и документиране на софтуерни архитектури.

3.1.1. Стъпка 1 – Избира се модул за декомпозиция

3.1.2. Стъпка 2 – Избира се архитектурни драйвери

3.1.3. Стъпка 3 – Избира се архитектурен модел (стратегия)

3.1.4. Стъпка 4 – Обособяват се подмодули

3.1.5. Стъпка 5 – Приписване на функционалност към подмодулите

3.1.6. Стъпка 6 – Създават се и други структури

3.1.7. Стъпка 7 – Верификация на декомпозицията

3.1.8. Стъпка 8 – Отиваме на стъпка 1 за някои подмодули

3.1.9. Стъпка 9 – Формират се екипи и се раздават задачи

3.1.10. Стъпка 10 – Създава се СКЕЛЕТ на системата

3.1.11. Стъпка 11 – Документиране на СА

Таблица – кой от какво се интересува (какви View-та)

3.2. Тактики (архитектурни решения)

Тактиката е архитектурно решение, чрез което се контролира резултата от даден **сценарий за качество**. Наборът от тактики се нарича **архитектурна стратегия** или **архитектурен модел**.

A. Тактики за готовност

a. Откриване на дефекти

- **Ping/echo** – един компонент, проверява друг компонент;
- **Heartbeat** – периодично излъчване на сигнал. В сигналът може да има някакви данни за състоянието на компонента или настъпили събития;
- **Exceptions (Изключения)** – обработка на изключения, за да не се чупи цялата система;

b. Отстраняване на дефекти

- **Voting** – на различни процесори работят различни процеси, които получават един и същ вход и би трябвало да генерират един и същ резултат. Ако има разлика в някой резултат централният процесор (voter) предприема някакви действия. Тази тактика обикнове се ползва за тестване на процесори;
- **Active Redundancy –** важните компоненти в системата са дублирани. Дублираните компоненти се подържат в едно и също състояние и резервният компонент е винаги готов да премине в състояние на активност. Downtime-ът е няколко милисекунди;

- **Passive Redundancy –** един активен и няколко резерви, които обаче се актуализират само ако основният компонент се дефектира. Downtime-ът е от няколко секунди, до няколко часа;
- **Резерва (Spare) –** поддръжка на резервни изчислителни мощности, които трябва да се инициализират и пуснат в действие при дефектиране на някой от компонентите;

c. Повторно въвеждане в употреба

- **Shadow Mode (Паралелна работа) –** преди да се въведе в употреба компонент, който е бил повреден, се пуска в shadow mode, за да се провери дали работи коректно.
- **Синхронизация на състоянието**
- **Checkpoints/Rollbacks**

d. Предотвратяване на дефекти

- **Transactions**
- **Removal from service (Извеждане от употреба) –** автоматично или ръчно
- **Process Monitoring –** посредством специален процес се следят основните процеси в системата. Ако даден процес дефектира, мониторинга може да го премахне, рестартира, дупликира и т.н.

B. Тактики за изменяемост

Директно (*променя се поведението*) и индиректно(*не се промня поведението, но се променя реализацията*) засегнати модули от дадена промяна.

a. Локализиране на промените (намалява броя на *директно* засегнатите модули)

- **Поддръжка на семантична свързаност (Cohesion)**
- **Очакване на промените (списък с евентуални промени)**
  Списъкът е основополагащ при декомпозицията на модули.
- **Ограничаване на възможните опции**
  При продуктовите линии се налага ограничение върху дадени очаквани промени с оглед на това да не се изменя тотално цялата продуктова линия.

b. Предотвратяване ефекта на вълната (да няма *индиректно* засегнати модули)

- **Скриване на информацията**
  - Декомпозиция на отговорността на даден елемент (система или конкретен модул) и възлагането ѝ **на по-малки елементи**, като при това част от информацията остава публична и част от нея се скрива.
  - **Публичната функционалност** и данни са достъпни посредством специално дефинирани за целта **интерфейси**.
- **Поддръжка на съществуващите интерфейси**
  - Добавяне на нов интерфейс в модула A
  - Wrapper на модула A
  - Добавя се нов модул A_REAL, а самият модул A става stub
- **Ограничаване на комуникацията (Loose Coupling)**
- **Използване на посредник**
  - **Синтаксис на данните (Repository)**

MVC и многослойните архитектури – преобразуване на данните от един слой в подходящ вид за друг слой)

- **Синтаксис на услугите**
  Ползват се посредници – чрез шаблоните *façade, bridge, mediator, strategy, proxy, factory.*
- **Идентификация на интерфейсите (Broker)**
  А предоставя няколко интерфейса. Б се свързва с брокер, който знае чий интерфейс да му предостави.
- **Местоположение на А (Name Server)**

c. Отлагане на свързването

- **Plug and Play**
  Възможност за добавяне и изключване на модули, докато системата работи.
- **Component Replacement**
  Възможност за добавяне и изключване на модули, при стартиране на системата.
- **Config files**
  При зареждане на системата или отделен модул от нея се взимат конкретни параметри.

C. Тактики за производителност

a. Намаляване на изискванията

- **Увеличаване на производителността на изчисленията (алгоритми/ хеширания)**
- **Намаляване на режийните (overhed) –** намаляване на действията, които не са пряко обвързани с конкретното събитие
- **Увеличаване на периода при повтарящи се събития**

b. Управление на ресурсите

- **Паралелна обработка**
- **Излишък на данни/ процеси (cache/ load-balancing)**
- **Включване на допълнителни ресурси**

c. Арбитраж на ресурсите

- **Scheduling –** приоритизиране на събитията

D. Тактики за сигурност

a. Устояване на атаки (ключалка на вратата)

- **Автентикация на потребители**
- **Нива на потребителски достъп**
- **Конфиденциалност на данните (криптиране)**
- **Ограничаване на достъпа (firewall, access control)**

b. Откриване на атаки (аларма)

- **Intrusion Detection Systems (IDS)**
  Интелигентно открива в реално време чрез анализ на трафика и БД реализирани атаки.

c. Възстановяване след атака (застраховка)

Тактиките за възстановяване се припокриват с тези за готовност, тъй като една атака може да се приеме като срив на системата.

E. Тактики за проверяемост

- **Разделяне на интерфейса от реализацията**
- **Вградени модули за мониторинг и журнал**

4. Архитектурни стилове и шаблони.

Архитектурният стил дефинира семейство от системи чрез използването на шаблон за структурна организация. Дефинирани са ограничения и начини за използването им.
Архитектурният шаблон (АШ) е описание на типове софтуерни елементи и връзки, заедно с ограничения относно тяхната употреба;

- В шаблоните няма описание на никаква функционалност освен тази, която служи за реализация на комуникационните протоколи;
- Шаблоните са полезни, тъй като те притежават добре известни качества. Архитектите най-често започват проектирането с избор на шаблон, от който да започнат;

4.1. Хранилище (Repository)

Подсистемите си комуникират единствено чрез хранилището. Сравнително независими са. Контролът върху поведението на системата може да се осъществява както от хранилището (при настъпването на някакво събитие в него), така и от подсистемите, които да освобождават ключове в хранилището и по този начин да дават зелена светлина на друга подсистема да ползва съответните ресурси.

4.2. MVC

Разделение на бизнес логиката от ГУИ. Разновидности на MVC

- Maintainability, flexibility

4.3. Клиент – Сървър

Сървърът не знае за клиента. Клиентът обработва и валидира действията на потребителя, след което праща заявки до сървърът. Сървърът се грижи за интегритет на данните.

4.4. P2P

- Прилича на Client-Server с разликата, че сървърът може да инициира контрол върху клиентите и освен това между клиентите може да се осъществява директна връзка.
- Структурирани, неструктурирани и централизирани p2p системи
- Distributed Hast Tables – fast resource discovery
- High Scalability
- Complex due to prevention algorithms of deadlocks

4.5. n-tier architecture

- Calls n-1 tier functionality
- Independent physical machines
- Maintainability, scalability, flexibility, availability

4.6. Pipes and Filters

Use the Pipes and Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes).

17. Софтуерна архитектура. Проектиране и документиране на софтуерни архитектури.

> Each filter exposes a very simple interface: it receives messages on the inbound pipe, processes the message, and publishes the results to the outbound pipe. The pipe connects one filter to the next, sending output messages from one filter to the next. Because all component use the same external interface they can be composed into different solutions by connecting the components to different pipes

5. Документиране на софтуерната архитектура. Предназначение на документацията. Основен принцип на документиране. Съдържание на документацията. Структура на документацията.

5.1. Предназначение на документацията

Документацията е от важност с оглед на това да може в бъдеще системата да бъде лесно поддържана, безпроблемно изменяна и ясно разказвана на клиенти, потребители.
Зле написаната документация често е по-лошият вариант от никаква документация, защото може да заблуди четящия.

5.2. Основен принцип на документацията

Документацията се състои от описание на различните структури и когато тя се пише се мисли най-вече **кой ще я чете**. Техноописецът трябва да се поставя на мястото на четящия. В документацията различните структури може да се групират по различни начини в зависимост от това, за кого конкретно е предназначена съответната документация.

5.3. Съдържание на документацията

- Първично представяне
- Описание на елементите и връзките
- Описание на обкръжението
- Описание на възможните вариации
- Архитектурна обосновка
- Терминологичен речник
- Допълнителна информация

5.4. Структура на документацията

- Каталог на структурите
- Шаблон за описание на структурите
- Кратко абстрактно описание на системата
- Защо така е проектирана СА