Европейски съюз

ОПАК. Експерти в действие

Европейски социален фонд
Инвестиции в хората

Проект „Повишаване квалификацията на служителите от администрацията на централно ниво чрез усъвършенстване на знанията и практическите им умения за управление на софтуерни ИТ проекти в съответствие със съвременните методологии“, осъществяван с финансовата подкрепа на Оперативна програма „Административен капацитет” (ОПАК), съфинансирана от Европейския съюз, чрез Европейския социален фонд”, съгласно Договор № К13-22-1/05.03.2014 г.

НАРЪЧНИК

ДЕЙНОСТ 2.

ПРОВЕЖДАНЕ НА ОБУЧЕНИЕ ЗА СОФТУЕРНИ АРХИТЕКТИ ЗА 33 СЛУЖИТЕЛИ НА ЦЕНТРАЛНАТА АДМИНИСТРАЦИЯ И ИЗДАВАНЕ НА СЕРТИФИКАТИ ЗА ПРОВЕДЕНОТО ОБУЧЕНИЕ

Изготвен в изпълнение на Договор № Д-37/11.12.2014 г.
между
МИНИСТЕРСТВО НА ТРАНСПОРТА, ИНФОРМАЦИОННИТЕ ТЕХНОЛОГИИ И СЪОБЩЕНИЯТА
и
„КОНСОРЦИУМ ИТ ОБУЧЕНИЯ 2015“ДЗЗД

New Horizons Computer Learning Centers

ciela norma

„КОНСОРЦИУМ ИТ ОБУЧЕНИЯ 2015“ ДЗЗД

София 1040, ж.к. Изток, бул. Драган Цанков 36, СТЦ Интерпред, блок А, ет.6; тел: 024210040; имейл: ittraining2015@newhorizons.bg;

Авторски колектив:

Симеон Ангелов

Редактор – проф. Владимир Димитров

Одобрил: Николай Пенев – ръководител на проекта

София, 2015 г.

Съдържание

Въведение

Този наръчник е част от учебните материали по софтуерна архитектура за служители на централната администрация. Той има за цел да подпомогне обучителя и обучавания при подготовката му по софтуерна архитектура.
Наръчникът няма за цел да бъде книга по софтуерна архитектура, а в структуриран вид в допълнение на проведения обучителен курс да даде препратки към места в глобалната мрежа, където да получите по-задълбочени познания.

Наръчникът се ползва както от обучаемите, така и от обучителите.

В наръчника се прави въведение в софтуерните архитектури и последователно надгражда умения за проектиране, дизайн, и поддръжка на системи за софтуерните архитектури. Дава повече обяснения и примери, което е предпоставка за по-добро и достъпно усвояване на материала

Речник на използваните термини

Термин	Превод/пояснение/препратка
Build	Процес на пакетиране на изпълним файл. Използва сорс кода и файловете за структуриране на изпълнимият файл. Крайният резултат, файлът, се разполага (внедрява) в сървър или друга машина, на който се изпълняват процедурите описани в сорс кода. Built процес играе ролята на предходна стъпка на Deploy процес
Stateless	Тип заявка, която не запазва състояние между различните транзакции. Всяка транзакция е напълно независима от данните, които се обработват. Пример: в уеб базираните системи при HTTP заявките при stateless такива нямат състояние в HTTP сесиите, които да се подава по между им
Database partitioning	Разделя голяма база данни или таблици на малки такива. С това се подобрява бързината на търсенето. Примерно такова разделяне на таблици е една голяма таблица да се раздели по редове на по 1000 всяка
Hybrid database	Различни база данни, работещи в една система. Има механизъм на синхронизация на данните и връзки по между им, така че да работят като една база данни – да бъдат прозрачни за клиента/системата които използват този вид база данни
Statefull	Тип заявка, която запазва състояние между различните транзакции. Транзакции са зависими по отношение на данните, които използват и пренасят по между си
Deploy	Процес на разполагане (внедряване) изпълним файл върху сървър или каква да е машина, на която може такъв файл да се изпълнява. Deploy е втора стъпка след build процеса
Релационна база данни	Вид база данни, в която има зависимости между различните таблици в нея. Тези зависимост са връзките между ключовете. Релационните база данни за системи, където има голям набор от данни не са удобни, поради бавната и тромава работа с тях и се заместват от нерелационни такива

Нерелационна база данни	Вид база данни, в които няма релации (зависимости) между единиците модели (документи и т.н.), който съдържат данните в базата данни. Типът данни, който се използва може да бъде графова, документна или ключ-стойност връзка
Cron job	Последователност от стъпки, описваща функционалност, което се изпълнява през определен период от време в точно определено такова. Примерно в база данни може да се конфигурира да се изпълнява изпращане на емайли до абонирани за поледни новини абонати един път седмично
Inversion of control	Описва дизайн на сорс код, в който контролът на стъпките описани чрез сорс кода се делегира от абстрактен тип модели. Това примерно за интерфейсите при обектно ориентираните програмните езици
Cluster	свързани компютърни центрове, работещи като една система. Идеята им е да разпределят натоваването в системата. С това примерно се подпомага horizontal partitioning при базата данни или хоризонталното мащабиране (horizontal scale) при сървърните web приложения
PoC	Proof Of Concept - това е сорс код в скелетен вид, който да докаже работеща имплементация
API	Application Programming Interface – това е набор от протоколи и правила за използване на софтуерни компоненти. За едно API се описват входните, изходните параметри, случаите му на използване и ограничения, ако има такива
Middleware	Софтуер, който свързва софтуерни компоненти или други софтуерни приложения. Това може да е слой в много слойната архитектура.
Multithreading	Многонишково приложение е такова, в което се манипулират повече от една нишка. Нишка или thread е последователност от изпълними стъпки. Многонишковият софтуер изпълнява конкурентно изпълними нишки

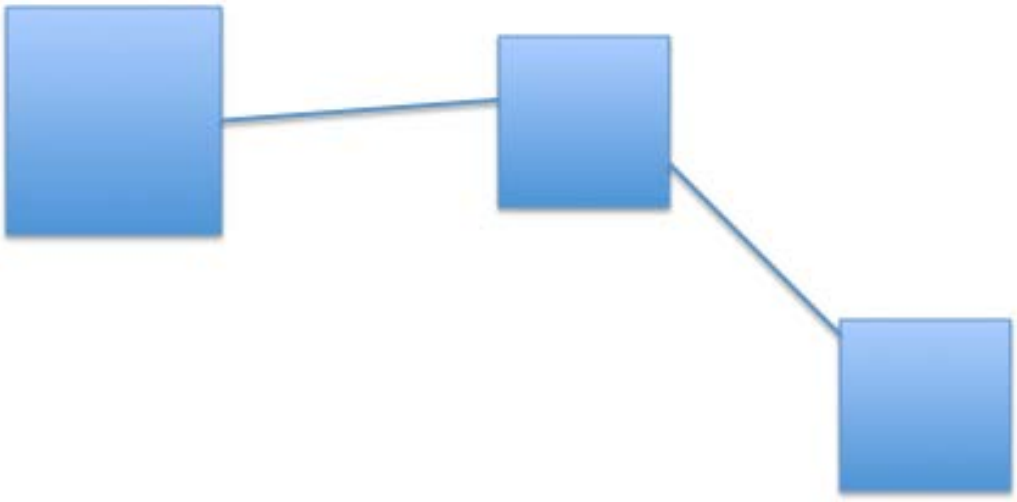
1. Модул 1 Какво е Софтуерна Архитектура

1.1. Нуждата от софтуерна архитектура

Нуждата от софтуерната архитектура е предизвикана от необходимостта за солидна основа при големите и сложни системи. Тя е гаранцията за дълъг живот на системата.
Ако не са предвидени различните сценарии и начини на използване на системата, общите проблеми и начините на справяне с тях, целите в дългосрочен план, то тогава софтуерният продукт е в риск. А това струва време и пари!
Разбиране на системата от всяко заинтересовано лице, нейното структуриране на елементите е друга необходимост. Софтуерната архитектура играе ролята на инструмент за комуникация, обосновка при вземане на решения, средство за анализ и развитие на системата.
Основно твърдение: Софтуерните архитектури са важни за успешното разработване на софтуерни системи. Има достатъчно и добре обобщени знания за софтуерните архитектури.
Софтуерните системи се изграждат за да удовлетворят бизнес целите на организациите. Архитектурата е мост между тези (често абстрактни) бизнес цели и крайната (конкретна) система.
Софтуерните архитектури могат да бъдат проектирани, анализирани, документирани и реализирани с определени техники, така че да бъдат постигнати целите на бизнеса.

1.2. Какво е софтуерната архитектура и какво не е

Нека разгледаме фиг. 1.1 и да се опитаме да извлечем информация за софтуерната архитектура на системата от нея.



Фиг. 1 Диаграма.

Можем ли да си отговорим на следните от въпроси от тази диаграма:

- Каква е връзката между тези квадратчета?
- Кой кого използва?
- Кога и по какъв начин си взаимодействат тези квадратчета?
- Кое каква функционалност представя/реализира?
- Колко са функциите?
- Задоволяват ли се нуждите на бизнеса?
- Какви са свойствата на тези „квадратчета“?

Софтуерната архитектура НЕ Е неразбираема, просто кутийки, без никаква информация за тях, без контекст, без бизнес цел и без заинтересовани от нея. Софтуерната архитектура на системата е множеството от структури, необходими за обсъждането й. Тя се състои от софтуерни елементи, отношенията между тях и свойствата им. Софтуерната архитектура е множество от структури, разглеждани в различен контекст. Тя е абстракция, необходима за разбиране на системата.
Съгласно Software Engineering Institute „Архитектура на софтуерната система е съвкупност от *структури*, представящи различните софтуерни елементи на системата, външно видимите им свойства и връзките между тях”
Други определения:

- Важни проектни решения, които трябва да се вземат рано и да не съдържат грешки.
- Модел на системата, използван от програмистите в проекта.
- Представяне структурата на системата от високо ниво.

Определението за софтуерна архитектура, съгласно третото издание на книгата „Software Architecture in Practice”, има три основни елемента за в описанието: „*Софтуерната архитектура е **набор от структури**, разглеждани в различен контекст в **единна абстракция**, които са необходими за да се представи системата, и тя включва **софтуерните елементи, връзките между тях, както и свойствата им***”. Основните моменти в определението са:

- набор от структури;
- абстракция;
- връзките и свойствата на софтуерните елементи.

Набор от структури

Една структура е архитектурна, ако описва системата и нейните свойства.

Видовете структури са:

1. **Модулите (modules)** разделят системата на единици за реализация. Те носят специфични изчислителни отговорности и са база за разпределение на работата между екипите от програмисти. Системата се декомпозира на модули. Модулната структура на системата е статична.
2. **Компонент-конектор (component-and-connector - C&C)** структурите представят системата по време на изпълнение. Те описват как елементите си взаимодействат за да постигнат функциите на системата. Компонентът е същност по време на изпълнение.
3. **Структурите на разпределението (allocation)** описват изображението на

софтуерните структури върху организационната среда, средата на разработка, инсталационната среда и средата на изпълнение на системата.

Архитектурата е абстракция

Архитектурата представя публичната част на системата, чрез нейните интерфейси. Интерфейсите скриват конкретната си реализация и предоставят контракт за взаимодействие с други елементи.
В този смисъл, архитектурата може да има множество реализации и за това е абстракция.
Всяка система има архитектура – документирана или не.
Архитектурата описва поведението на елементите по време на изпълнение.
Не всички архитектури са добри по отношение на бизнес целите.

1.3. Архитектурни структури и изгледи

Структурата (structure) е самото множество от елементи, така както те съществуват в софтуера или хардуера.
Изгледът (view) е представяне на кохерентно множество от архитектурни елементи, така както са записани или четени от заинтересованите в системата. Той представя множеството от елементи и отношенията между тях.
Архитектурните структури са:

- модулните структури;
- структурите компонент-конектор;
- структурите на разпределението.

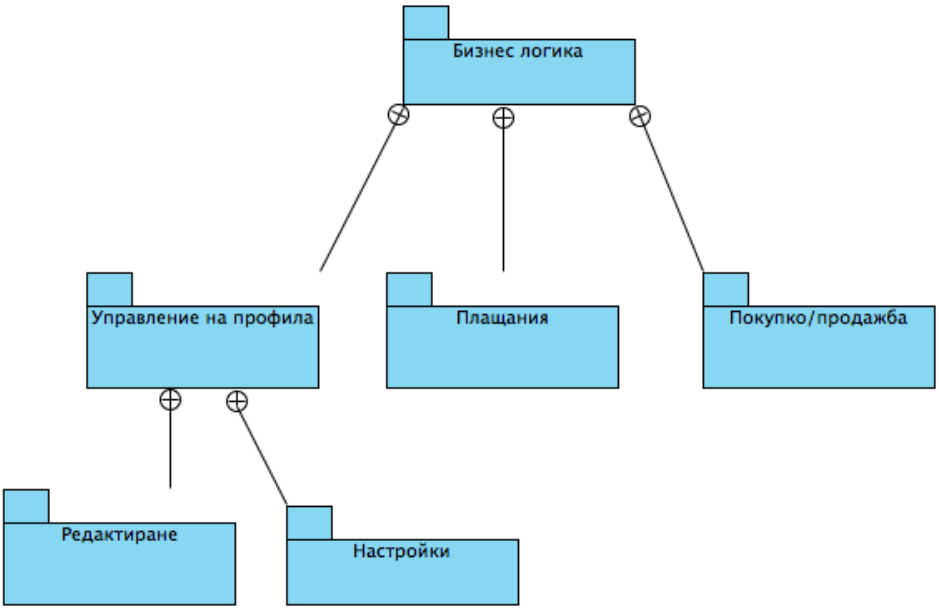
Архитектурни структури – модули

Модулите дефинират структурата на системата като съдържание на код, единици работа за разработка и т.н.
Отговарят на въпроси като:

- Каква функционалност модулът реализира?
- Кои други елементи може да го използват/извикват?
- Кои други модули зависят от него?

1. Декомпозиция на модулите

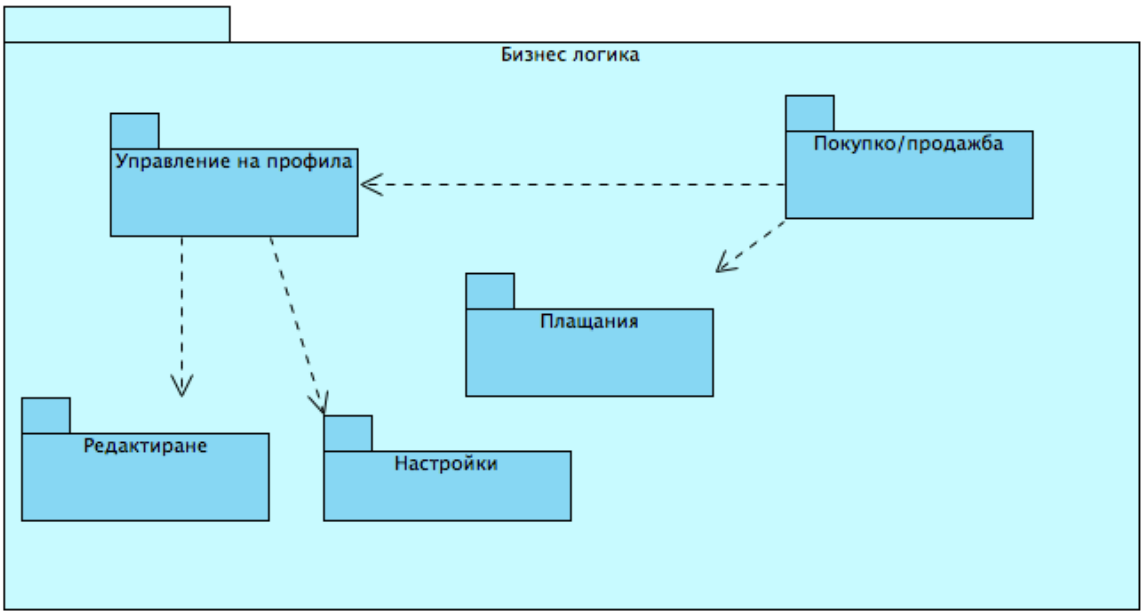
Декомпозицията на модулите (module decomposition) е асоциация между модулите от вида „X е подмодул на Y“.
Декомпозицията на модулите определя в голяма степен възможността за промяна, като обособява логически свързаните функционалности на едно място.
Много често, декомпозицията по модули служи и като основа на разпределението на работата между екипите на разработка.



Фиг. 2 Декомпозиция на модулите.

2. Използване на модулите

Използването (use) на модулите е асоциация между модулите са от вида „X използва Y“.
Ако има нужда от по-детайлно описание, може асоциациите да са насочени към конкретен интерфейс или ресурс на модула.
Структурата за използване на модулите дава възможност за лесно добавяне на нова функционалност, за обособяване на самостоятелни подмножества от функционалности, и позволява разработка на системата с нарастване.



Фиг. 3 Използване на модулите.

3. Структура на слоевете

Частен случай на структурата на използване на модули е структурата на слоевете. Тук, върху използването са наложени строги правила, които обособяват слоевете. Модулите от слой N могат да използват само услугите на модулите от слой N-1. Слоевете често се реализират като виртуални машини или подсистеми, които скриват детайлите на работата си от по-горния слой.
Структурата позволява без особени сътресения да бъде подменен цял един слой (напр. да се смени СУБД).

Фиг. 3 Структура на слоевете.

Архитектурни структури – структура компонент-конектор

В структурата компонент-конектор елементите са ***компонентите***, по време на изпълнението (т.е. основните изчислителни процеси) и ***конекторите***, които са средствата за комуникация между компонентите.

Някои от въпросите, на които отговарят тези структури:

- Кои са основните изчислителни процеси и как те си взаимодействат?
- Кои са основните споделени ресурси?
- Как се изменят данните в системата?
- Кои части от системата могат да работят паралелно?
- Как се променя структурата на системата докато тя работи?

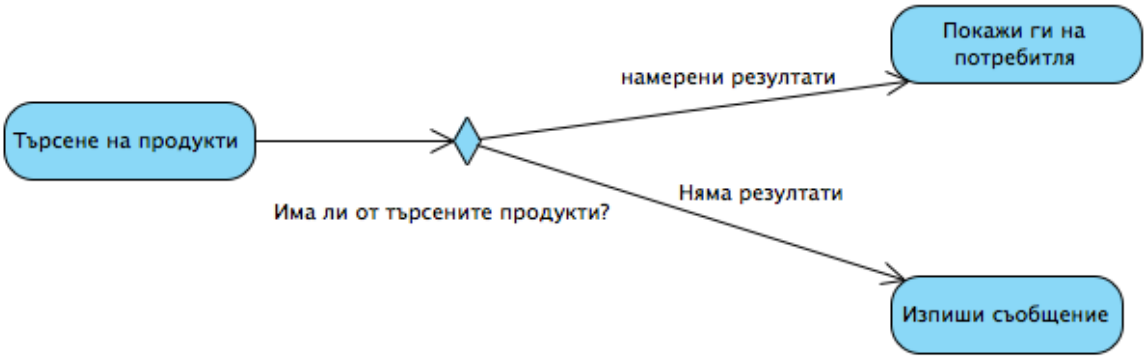
Структура на конкурентността (компонент-конектор)

В структурата на конкурентостта, елементите са нишките, изпълнявани в системата (компоненти) и комуникационни, синхронизационни или блокиращи механизми между тях (конектори).

Тези структури показват как нишките могат да се изпълняват конкурентно.

Връзките показват свързаността между компонентите чрез конекторите.

Структурата е полезна за анализ на изпълнението и надеждността.



Фиг. 4 Пример на структура на конкурентостта.

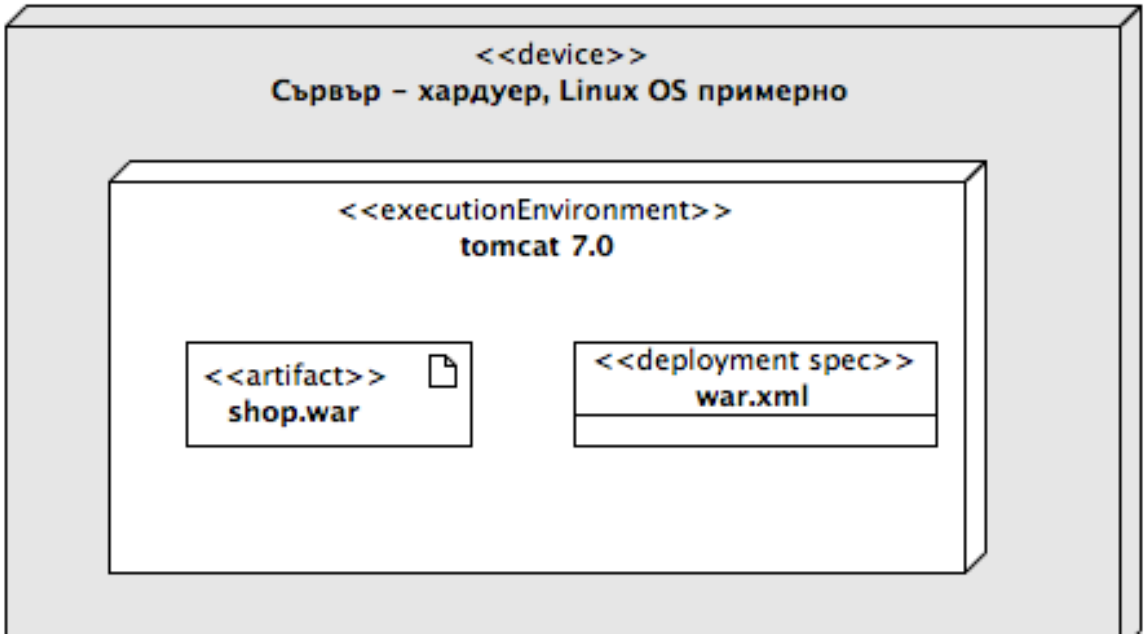
Архитектурни структури – структура на разпределението

Структурата на разпределението показва как елементите компонент-конектор или модулите се изобразяват върху несофтуерните елементи като хардуер, екипи и файлови системи.
Помага да се отговори на въпроси като:

- Коя част от системата върху кой процесор се изпълнява?
- В коя база от данни се съхраняват данните?
- Кой екип, коя част от софтуера, разработва?

Разпределение на работата (структура на разпределението)

Отразява кой екип, за кой модул носи отговорност, т.е. за реализацията на модула. Елементите са модули и екипи, а връзките са кой модул от кой екип се разработва. Под „кой екип“ не се има предвид конкретен списък от хора, а по-скоро група от хора с подходящ опит, знания и умения.



Фиг. 5 Пример за структура на разгръщането (структура на разпределението).

Кои структури да използваме?

Има голям набор от структури. Всяка структура е с различна цел и е предназначена за определена група от заинтересовани лица. Важно е да се избере такъв набор от структури, с който да се представи по възможно най-ясен начин софтуерната архитектура пред заинтересованите.

1.4. Архитектурни образци

В архитектурните образци, софтуерните елементи са композирани по начин, който решава определен проблем.
Софтуерните образци са открити в практиката. Те са добри практики решаващи даден проблем. Софтуерните образци са добре документирани и илюстрирани с множество примери.
Архитектурният образец е решение на даден проблем в даден контекст, което може многократно да се използва.
Архитектурният образец е достатъчно абстрактен за да има многократна употреба. Той отразява само елементите, взаимодействията и последствията.
Примери са:

- Образец на слоевете (Layered pattern): когато използването на елементите в софтуера е само в една посока, където всеки слой реализира силно свързана логика (достъп и работа с база от данни, бизнес логика и т.н.).
- Образец клиент-сървър (Client-Server pattern): компонентите са клиента и сървъра. Конекторите са протоколите и съобщенията които компонентите си обменят.

1.5. Видове архитектури

Архитектурите биват:

- системна архитектура (system architecture);
- архитектура на предприятието (enterprise architecture);
- приложна архитектура (application architecture).

Системната архитектура е представяне на системата, което изобразява функционалността в хардуерни и софтуерни елементи, изобразява софтуерната архитектура в хардуерната архитектура, и разглежда взаимодействието на хората с тези компоненти. Системната архитектура разглежда цялостната система, включвайки хардуер, софтуер и хора.
Архитектурата на предприятието е описание на структурата и поведението на процесите, информационните потоци, персонала и организационните единици в организацията, подравнени по основните цели и стратегията на организацията. Софтуерът е една от темите на архитектурата на предприятието. Другите две теми са как софтуерът се използва от хората за изпълнение на бизнес процесите и как стандартите определят изчислителната среда.
Архитектурата на предприятието представя софтуерната системата чрез различни изгледи. Съдържа модели, структури и поведение. Налага архитектурни принципи и стандарти. Има пряка връзка с архитектурата на бизнеса.
Видовете архитекти са:

- приложен архитект (application architect);
- архитект на решението (solution architect);
- архитект на предприятието (enterprise architect);
- архитект на облака (cloud architect);
- инфраструктурен архитект (infrastructure architect);
- системен архитект (system architect).

Какво прави една архитектура добра?

Няма критерии, по които да се определи дали една архитектура е добра или не. Но въпреки това, има някои признаци (препоръки), по които може да се каже дали дадена софтуерна архитектура е добра.

Например, ориентираната към услуги архитектура е подходяща за големи бизнес-към-бизнес решения, но не и за малки сайтове за електронна търговия или пък за сложни авиационни системи.

Ако препоръките не са изпълнени, това е предупреждение, че начинанието може и да не успее.

Дали една софтуерна архитектура е добра или лоша, може да се оцени само по отношение на целите ѝ.

Фиг. 6 Връзка между архитектура, архитект и обхват.

1.6. Препоръки за добра архитектура

1. Софтуерната архитектура трябва да е разработена от един архитект или от екип архитекти с изявен лидер в екипа.
2. Софтуерната архитектура трябва да е изградена въз основа на добре дефинирани изисквания и списък с приоритети на качествените характеристики.
3. Софтуерната архитектура трябва да бъде добре документирана с подходящите изгледи (views), предназначени за различните заинтересовани лица.
4. В развитието на софтуерната архитектура трябва да се внимава за поддръжка на качествените и бизнес характеристиките.
5. Софтуерната архитектура трябва да има добре дефинирани модули, всеки от които реализира определена функционалност.
6. Качествените атрибути трябва да се постигат чрез добре известни тактики и образци.
7. Архитектурата никога не трябва да зависи от конкретна версия на инструмент или софтуер.
8. Модулите, които произвеждат данни, трябва да са отделени от модулите, които консумират данни.
9. Всеки процес трябва да се реализира така, че да може да се сменя процесора, върху който той се изпълнява.

Обзор

Софтуерната архитектура е множество от структури, с които може да се опише една система. Тя представя елементите, с които е реализирана и връзки между тях.

Структурата е набор от елементи и взаимодействия им.

Гледката е начин на представяне на елементите на софтуерната система, предназначена за определена група заинтересовани лица.

Софтуерната архитектура отразява изискванията на бизнеса, с подходящо подбрани атрибути за качество на изискванията. Тя използва подбрани архитектурни образци.

2. Модул 2 Защо е важна софтуерната архитектура?

Софтуерната архитектура представя най-ранните проектни решения.
Софтуерната архитектура оказва влияние върху организационната структура.
Софтуерната архитектура подобрява прогнозирането на разходите и графика на изпълнение.
Софтуерната архитектура задава модел за многократна употреба.
Софтуерната архитектура позволява обединяването на независимо разработени компоненти.
Софтуерната архитектура е основа за обучението и за изграждането на общ терминологичен речник.

Въпросите пред една софтуерна архитектура са:

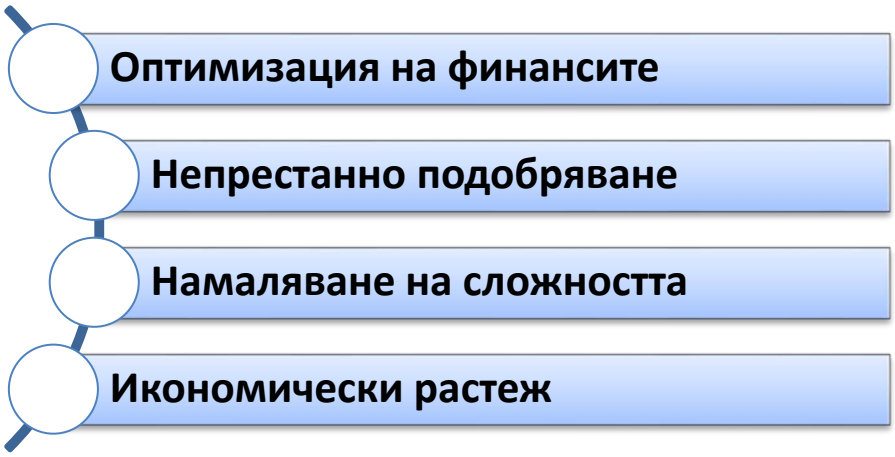
- Защо една добре описана софтуерна архитектура е добра?
- Какви ползи има от нея?
- Каква добавена стойност носи, както от бизнес, така и от техническа гледна точка?

2.1. Добавената стойност от софтуерната архитектура

Добавената стойност от софтуерната архитектура се изразява в:

- предоставената възможност за оптимизация на финансите;
- откриване на пътища за непрестанно подобряване;
- намаляване на сложността на системата;
- стимулиране на икономическия растеж.

Софтуерната архитектура свързва ***мисията на бизнеса***, неговата ***стратегия*** с процесите на организацията, така че да бъдат изпълнени ***ефективно, устойчиво*** и ***адаптивно*** настоящите и бъдещите нужди на организацията.



Фиг. 7 Добавената стойност от софтуерната архитектура.

Има 10 основни точки, които дават ясна представа за стойността на архитектурата от техническа гледна точка, които са разгледани в следващите раздели.

2.2. Поглъщане или постигане на атрибутите за качество на софтуерната архитектура

Винаги при проектиране на софтуерната архитектура, трябва да се има предвид кои са основните атрибути за качество и кои не.
Например, ако е необходимо висока производителност (performance), се търси конкурентно изпълнение на процесите. Докато за лесни изменения (modifiability) фокусът е върху това: кой модул какви отговорности носи и как модулите си взаимодействат.
С добре зададени атрибути на качеството, още в началото на проекта, може да се предвидят стратегии за постигането им и постепенната им реализация.

2.3. Разбиране и управление на промените

Изменяемостта (modifiability) е основополагащ атрибут на качество, особено в съвременните системи.
Най-вече, програмистите си имат работа с изменение на наличен софтуерен продукт. Още в началото, докато се прави проекта, трябва да се анализират детайлно кои части от системата, за какво ще отговарят, за да се постигне силна кохезия и слаба зависимост между елементите.
Така, измененията на софтуера ще бъдат ефективни и бързи.

2.4. Предвиждане на атрибутите за качество на системата

Колкото по-рано се открие проблем в проекта на софтуерната система, толкова по-лесно и евтино е преодоляването му.
От значение е не само да се изберат водещите атрибути за качество, но и да се има предвид кои други биха спомогнали за качеството на продукта.
Трябва да се следи развитието на софтуера и дали се постигат атрибутите за качество, също така и кои други атрибути биха им спомогнали, особено свързаните с тях.

2.5. Подобряване на комуникацията между заинтересованите страни

Заинтересованите лица (stakeholders) са всички, които имат отношение към създаването и използването на софтуерната система:

- собствениците;
- ръководителите;
- специалистите по продажбите;
- ръководителят на проекта;
- разработчиците;
- екипът по поддръжка;
- различните групи при клиента;
- крайните потребители и т.н.

Интересите на заинтересованите най-често влизат в противоречие.

Обикновено, архитектът е в безизходица – каквото и да предприеме, все някой от

заинтересованите ще е недоволен.

Архитектурата предоставя общ език за всички заинтересовани лица. Тя дава възможност да се гледа на системата в различен контекст (гледка).

Ролята на архитекта е да търси компромис между заинтересованите лица, за да бъдат техните интереси отразени в спецификацията на изискванията!

2.6. Задава ограничения върху реализацията

Софтуерната архитектура определя реализация, водена от проекта й.
Налага ограничения в използваните структури.
Елементите взаимодействат помежду си по строго определени правила.

2.7. Софтуерната архитектура представя най-ранните проектни решения

Какви могат да са тези ранни решения, засягащи софтуерната архитектура?
Например:

- Системата на един процесор ли ще се изпълнява или на няколко процесора?
- Софтуерът ще е разделен ли на слоеве? Ако да, на колко слоя? Кой слой какво ще прави?
- Компонентите как ще си взаимодействат? Синхронно или асинхронно?
- Системата зависи ли от други системи? Как те ще се интегрират?
- Ще се криптира ли информацията в системата?
- Кой протокол ще се използва за комуникация?

2.8. Влияние върху структурата на организацията

Разпределението на отговорностите е в основата на правилното разпределение (организация) на работа и на комуникацията между екипите.
Организацията се основава на целите, по които се създава системата (изискванията ги отразяват най-пълно).
Организационна структура допринася за правилно управление на проекта, за планиране и за оценяване на извършената работа.

2.9. Подобряване на прогнозиране на разходите и графика

Разходите и времето са важен инструмент за управлението на проекта.
Архитектът взима участие в оценката на разходите и времето през целият му жизнен цикъл на проекта.
Оценката е точна само ако изискванията към системата са прегледани детайлно и са валидирани. Резултатът трябва да е по-ясен и точен обхват на работата във времето.

2.10. Софтуерната архитектура може да даде модел с многократна употреба

Ако многократното използване на код е просто облага, то архитектурата

многократно може да се използва за системите с подобни изисквания.
Не само код, но и инфраструктура, софтуерни решения, инструменти и най-вече опит могат многократно да се използват.
Софтуерната продуктова линия (Software product line) е множество от софтуерни системи, реализирани с подобни инструменти. Те са мощно средство за висока производителност, ниска цена и доказано качество на продукта.

2.11. Обединяване на независимо разработени компоненти

Архитектурата се фокусира върху композирането на независими един от друг софтуерни елементи.
Композицията позволява елементите да се използват и в други подобни системи (продуктова линия).
За това още в самото начало, при проектирането на архитектурата, се преглежда, какво има на пазара, и кое би могло да се използва/интегрира.

2.12. Софтуерната архитектура е основа за обучението и създаването на общ тълковен речник

Архитектурата описва основните елементи и комуникация между тях. Тя е база за запознаване със системата на новите членове в проекта.
Тя създава речник, с който може лесно да общуват членовете на екипа (или заинтересованите лица).

2.13. Обзор

Софтуерната архитектура е значима за широк спектър от технически и нетехнически (бизнес) решения.
Правилното разбиране и прилагане на софтуерната архитектура в процеса на разработка, допринася за ефективност на работата, както на разработчиците, така и на бизнеса.

3. Модул 3 Контекст на софтуерната архитектура

В този модул се разглежда софтуерната архитектура в различен контекст: какви дейности се извършват, какво влияние има върху контекста и каква роля играе в контекста.
Контекстът може да бъде:

- техническият контекст;
- жизненият цикъл на проекта;
- бизнесът;
- професионалното развитие.

3.1. Архитектурата в технически контекст

Има три основни влияния на архитектурата върху техническия контекст:

- Софтуерната архитектура може да стимулира постигането на атрибутите за качество;
- От архитектурата на една система, може да се предвидят значими аспекти на качеството на системата;
- С архитектурата много по-лесно и ефективно се обсъждат и управляват промените.

Архитектурата и атрибутите за качество

Ролята на архитектурата в технически аспект, се свързва със задаването на атрибутите за качество на системата.
За да притежава една система изискваните качествени характеристики, те трябва да бъдат заложени по време на проектирането, но също така при разработката и внедряването.
Всяко качество се определя от това какво се иска (***ако***) и какво трябва да се направи за да се постигне (***то***).
Ако се изисква висока производителност (performance), ***то*** фокусът е върху времето, необходимо да се изпълнение на заявките, споделянето на ресурсите, логиката на конкуренция между нишките, разпределянето на функционалността по компонентите.
Ако се изисква наличност (availability), ***то*** се разглеждат сценарии на справяне при провал на част от системата или на реакция при различни видове грешки.
Ако се изисква използваемост (usability), ***то*** решенията по отношение на архитектурата, включват предоставянето на полезна функционалност от вида на отказ (cancel), възстановяване (undo), използване (re-use) на предишно въведени данни и т.н.
Ако се изисква изменяемост (modifiability), ***то*** тогава фокусът е върху декомпозиция на модули и тяхната взаимовръзка.

Архитектурата и техническата инфраструктура и среда

Техническата среда и инфраструктурата влияят върху проектирането на архитектурата.
Съвременните технологии влияят чрез:

- индустриалните стандарти;
- добрите практики;
- преобладаващи инженерни техники.

Влияние имат и добрите архитектурни стилове, като уеб-базираните, ориентираните към услуги архитектури, облаците и т.н.

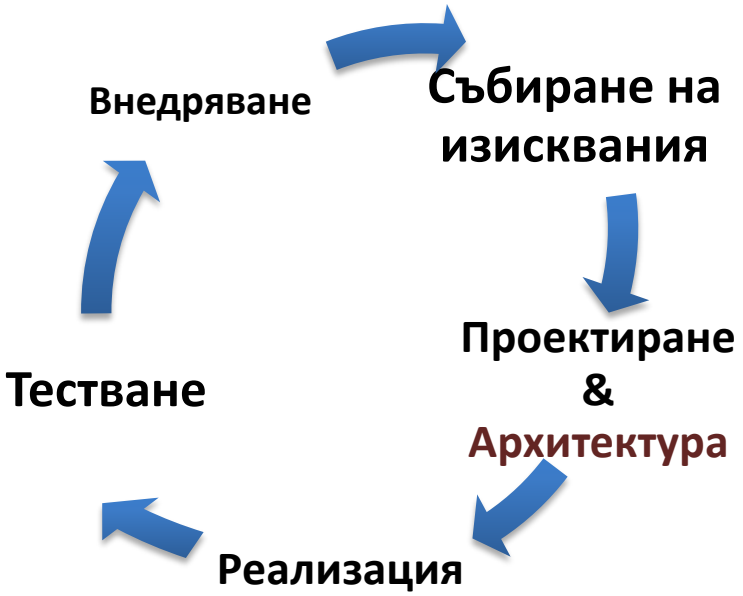
3.2. Архитектурата в жизнен цикъл на проекта

Процесите за разработка на софтуер са стандартизирани подходи за моделиране и програмиране.
Архитектурата играе съществена роля в процесите за разработка.
Има 3 широко разпространени процеси на разработка:

- водопад (waterfall);

- итеративен (iterative);
- подвижен (agile).

Фиг. 8 Водопад.

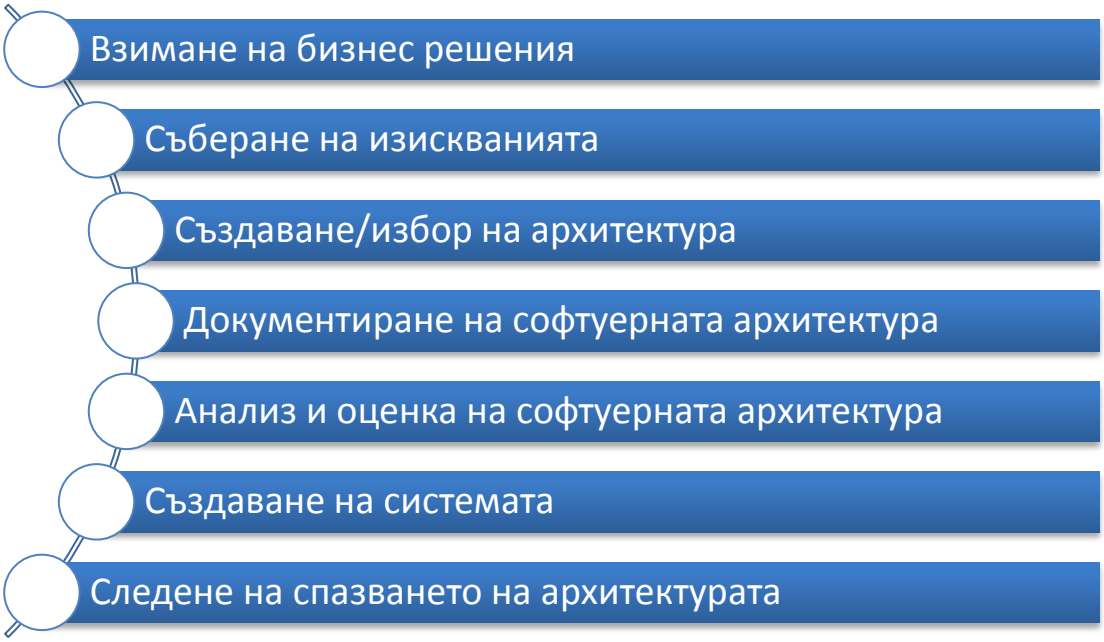


Фиг. 9 Итеративен.

Фиг. 10 Подвижен.

3.3. Дейности свързани с архитектурата

Независимо от това какъв е процесът на разработка, архитектурата изисква определена поредица от действия.
Процесът на разработка определя до каква степен се набляга върху тези дейности.



Фиг. 11. Дейности свързани с архитектурата.

Вземане на бизнес решения

За да се вземе решение за създаване на дадена система, трябва да се отговори на въпросите:

- Каква е целта на системата?
- Колко ще струва?
- За колко време ще се реализира?
- В каква среда ще работи (интерфейси с други системи)?
- Какви ограничения се налагат върху системата?

По тези въпроси архитектът следва да вземе отношение. Ако той не участва във вземането на бизнес решението, вероятността за провал се увеличава.

Събиране на изискванията

Изискванията (функционални и нефункционални) обуславят софтуерната архитектура.
Изискванията трябва да се задават по възможно най-недвусмислен начин.
Ако архитектът участва в задаването на изискванията, вероятността да се създаде система, която отговаря на поставените бизнес цели е по-голяма.

Създаване или избор на архитектура

Създаването или изборът на архитектура е същинската работа на архитекта.
Съществено тук, е че успешен проект и разработка, могат да се изградят само ако е налице идейна цялост, а такава може да се постигне само с последователен и подреден мисловен процес на специално избран(и) архитект(и).

Документиране на софтуерната архитектура

Документирането на софтуерната архитектура е втората част от същинската работа на архитекта.
И най-добрата архитектура е безполезна, ако тя не бъде представена на всички заинтересовани лица по подходящия начин.
Нюансът тук, е че формата, под която трябва да бъде представена софтуерната архитектура, зависи от спецификата на заинтересованите лица.

Анализ и оценка на софтуерната архитектура

Във всеки процес на проектиране има по няколко варианта, които следва да се оценят и анализират за да се избере най-добрия.
Архитектурите се оценяват, както по отношение на постигането на изискванията, така и по отношение на финансовите параметри.

Създаване на системата

Ролята на архитекта, по време на създаването (implementation) на системата, е да следи дали се спазват предписанията на софтуерната архитектура.
Това, че има прекрасна, добре документирана и представена архитектура е добре, но

ако хората, които правят системата не я следват, ефектът е нулев.

Следене за съответствие

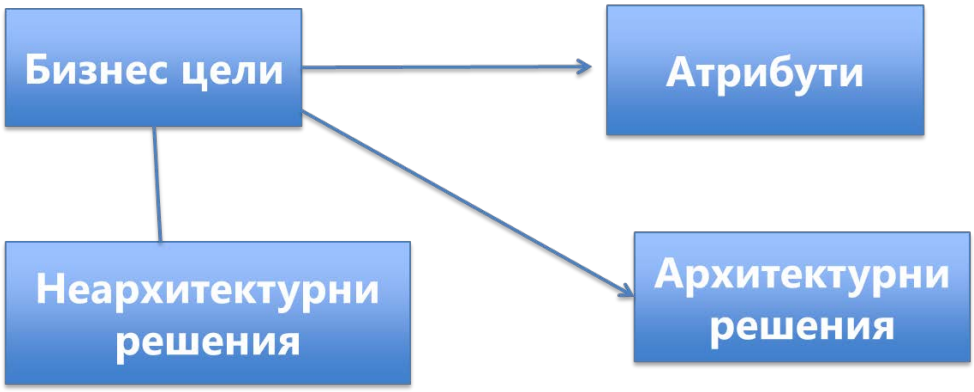
След като системата бъда разработена и премине към поддръжка, архитектът трябва да следи за съответствието между софтуерната архитектура и системата. По време на поддръжката се налагат промени, но тяхната реализация следва е в съответствие с принципите на архитектурата. От своя страна, софтуерната архитектура също трябва да се адаптира към промените.

3.4. Архитектурата в бизнес контекст

Архитектурите и системите не се правят безцелно. Те имат определена бизнес цел. Организациите, обикновено, търсят печалба, по-добра позиция на пазара, по-висок рейтинг. За да постигнат това, архитектите трябва да са наясно бизнес целите на организацията.
Много от бизнес целите се дефинират за една система като изисквания за постигане на атрибути качеството.
Всеки атрибут на качество трябва да дава отговор и в бизнес контекста:

> Например: "Защо е необходимо бързо да се отговаря на заявка?", то той би бил: "С това продуктът ще се различава от този на конкуренцията".

Разбира се, не всяка бизнес цел води до архитектурно решение или до постигането на атрибут на качество.
Обобщено, всяка бизнес цел в архитектурата води до изискване към атрибут на качество, към архитектурно решение или до неархитектурно решение.



Фиг. 12 Архитектура и бизнес цели.

Архитектурата и организацията

Бързото и ефективно постигане на резултати влияе и на това как организацията в информационно-технологичен контекст е структурирана.
В това дали да се правят инвестиции за произвеждането на решения (software solutions), дали се набляга на продуктови линии (product lines), или на инфраструктурни решения с използването на облачни системи (cloud) или локални сървъри.
Начинът на организация на софтуерната разработка може или да благоприятства, или не работата по архитектурата.

3.5. Архитектурата и професионализма

Архитектът трябва да има не само чисто технически знания.
Той трябва да има комуникативни умения, за да може да обясни на различни заинтересовани лица, в различни аспекти, взетите на техническите решения.
Той трябва да има обширен опит: да е използвал различни архитектурни подходи, като 3-слоева архитектура или пък клиент-сървър, или стилове като публикуване–абонамент; да знае недостатъците и предимствата на всеки от тях и да съумява да преценява използването им.
Той винаги трябва да се интересува и изучава новите технологии, стилове и добри практики.

3.6. За заинтересованите страни

Заинтересовани лица (*Stakeholders*) са всички, които имат отношение към създаването на софтуерната система: собствениците, ръководството, специалистите по продажби, ръководителя на проекта, разработчиците, екипът по поддръжка, различни групи на клиента, крайните потребители и т.н.
Всички заинтересоване имат разнопосочни интереси, например:

- Системата да се държи по определен начин.
- Системата да работи добре върху определен хардуер.
- Системата да може лесно да се променя.
- Системата да стане бързо.
- Системата да е евтина.
- Системата да я правят хора с конкретни умения.
- Системата да е многофункционална.
- и т.н.

Интересите на заинтересованите лица, най-често, са в противоречие.

Обикновено, архитектът е в безизходна позиция: какъвто и ход да предприеме, все някой от заинтересованите ще е недоволен.

Ролята на архитекта е да търси и намира компромис между заинтересованите лица, за да могат конкретните интереси да бъдат отразени в спецификацията на изискванията!

3.7. Влияние на архитектурата и върху нея

Влияние върху архитектурата

В много редки случаи изискванията, породени от бизнес целите, са напълно разбрани, обяснени и документирани.
Това води до *конфликти* между различните заинтересовани лица, които трябва да бъдат преодолени.
За целта архитектът трябва да разбере *същността, източниците и приоритетите на различните ограничения* и трябва да управлява нуждите и очакванията на

заинтересованите лица.
Крайната цел е заинтересованите лица да приближат позициите си, така че да се намери пресечна точка между противоречивите на пръв поглед изисквания.

Междинното звено е архитекта

Всичко това означава, че за да бъде един архитект успешен, той се нуждае от качества като:

- Отлично познаване на технологиите;
- Отлично аналитично мислене;
- Комуникативност, дипломатичност и умение за убеждаване, и въобще за водене на преговори.

Влияние на архитектурата

Веднъж създадена, софтуерната архитектура, от своя страна, влияе върху същите тези фактори, които са определили създаването ѝ: организацията, нейните цели, опитът на архитекта, изискванията към бъдещите системи, в някои случаи дори и върху технологичната среда.

4. Модул 4 Атрибути на качеството

4.1. Увод

Качеството на една система се определя от фактори, различни от функционалността ѝ.
Качеството на системата главно се определя от способностите и поведението на системата.
Атрибути като мащабируемост, изменяемост, сигурност и т.н., не са заложени в реализацията на функционалните изисквания.
Атрибутът на качеството е измеримо или подлежащо на тестване свойство на системата, което определя доколко системата задоволява нуждите на заинтересованите лица.

4.2. Архитектура и изисквания

Фиг. 13 Архитектура и изисквания.

Функционалните изисквания определят какво ще прави системата. Всяка система извършва работата, за която е предназначена.
Функционалните изисквания се постигат чрез разпределяне на отговорностите в различните модули на системата.
Изискванията на атрибутите за качество са квалифициране на функционалните изисквания в определен контекст, например:

- Колко бързо системата отговаря на заявка?
- Как реагира на грешка?
- Колко е гъвкава системата при изменение на функционално изискване?

Изискванията на атрибутите за качество се постигат чрез проектни решения и структури в системата, чрез поведението и комуникацията между елементите.
Ограничението е проектно решение, което вече е направено. Над ограничението надграждаме. Ограничението може да бъде избрана архитектура, избран език за програмиране, функционален стек, рамка и т.н. Ограниченията са задължителни. Ограниченият трябва да се съчетават с проектните решения, за да бъдат постигнати.

4.3. Функционалност

Системата върши работата, за която е предназначена.

Функционалността не определя архитектурата.

Функционалността може да се реализира както в един **монолитен блок**, така и чрез **модули, слоеве, класове, сървиси/услуги, процеси** и т.н. Вторият вариант е за предпочитане, тъй като софтуерът става по-разбираем и ефикасен!

Софтуерната архитектура определя структурата на системата в контекста на нефункционалните изисквания (атрибутите на качеството).

Функционалността е важна за архитектурата доколко тя взаимодейства с атрибути на качеството?

4.4. Атрибути на качеството

Атрибутите на качеството се залагат във всичките фази на проекта – от проектиране до внедряване.

Нито едно качество не зависи изцяло от проектирането или пък от внедряването.

Постигането на задоволителни резултати е въпрос на правилно разбиране, както и на цялостната картина – софтуерната архитектура, така и в детайлите на разработка.

Атрибутите на качеството не се разглеждат в изолация.

Постигането на едни атрибути може да окаже положително или отрицателно влияние върху други.

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+		+		
Integrity		-			-				-		-	-
Interoperability		-	+	-		-						
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-				-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

Фиг. 14 Взаимно влияние между атрибутите за качество.

Атрибутите на качеството се разделят на следните три групи:

- Технологична група: надеждност, изменяемост, производителност, сигурност и т.н.
- Бизнес група: време за излизане на продукта на пазара и т.н.
- Архитектурна група: те са присъщи на цялата система. Влияят косвено на цялостната система и на останалите качества.

4.5. Сценарии за качество

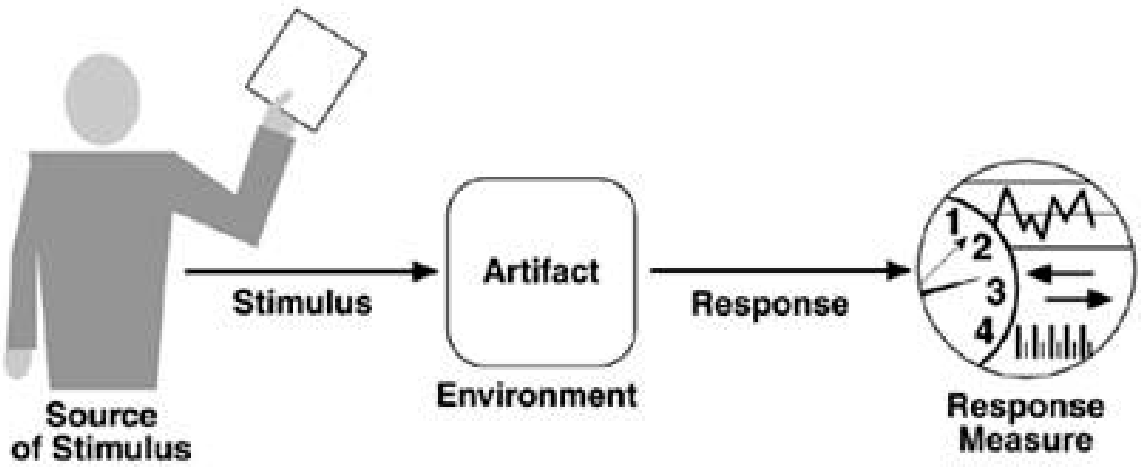
Сценарият е форма за специфициране особеностите на атрибутите за качество.

Сценарият за качество е специфично изискване към поведението на системата – атрибут на качеството.

Потребителският случай при дефинирането на бизнес сценариите е аналог на сценариите на качество.

Всеки сценарий за качество се характеризира с 6 елемента:

- Стимул (Stimulus)
- Източник на стимула (Source of Stimulus)
- Артефакт (Artifact)
- Среда (Environment)
- Отговор (Response)
- Замерване на отговора (Response Measure)



Фиг. 15 Общ сценарии за качество.

Сценарий за надеждност:

- По време на нормална експлоатация на системата ... (среда)
- ... външен източник (източник на стимула)
- ... изпраща неочаквано съобщение X ... (стимул)
- ... което се получава от процеса X ... (артефакт)
- ... Процесът трябва да информира оператора и да продължава нататък ... (отговор)
- ... без прекъсване ... (замерване на отговора)

Сценарий за изменяемост:

- Преди пускането на системата в експлоатация (среда)

- ... клиентът ... (източник на стимула)
- ... желае промяна на създаване на профил ... (стимул)
- ... За целта се променя модула за менажиране на профили (артефакт)
- .. Това не трябва да предизвиква никакви странични ефекти (отговор)
- Времето за извършване на промяната е 16 часа (замерване на отговора)

4.6. Постигане на атрибутите на качество чрез тактики

Защо един проект притежава висока сигурност, а друг висока производителност?

Постигането на тези качества е въпрос на фундаментални архитектурни решения – тактиките.

Тактиката е архитектурно решение, чрез което се контролира резултата на даден сценарии за качество.

Наборът от конкретни тактики се нарича архитектурна стратегия.

Фиг. 16 Постигане на атрибутите на качество чрез тактики.

4.7. Проектиране на атрибути на качеството

Архитектурата е набор от проектни решения.
Тактиките се описват чрез:

- разпределение на отговорностите;
- модел на координация;
- модел на данните;
- управление на ресурсите;
- връзка между архитектурни елементи;

- избор на технология.

Разпределение на отговорности

Идентифицират се основните, водещи отговорности (системни функции, инфраструктура ...)
Разпределение им става по модули, процеси, и т.н.
Стратегии, които се използват за разпределение на отговорностите са функционална декомпозиция, моделиране на процеси и т.н.

Модел на координация
Моделът на координация е механизъм за интеграция. Трябва да се определят моделите, които се нуждаят от координация помежду си, а също така свойствата и мерките на координацията – време, точност, честота.
Моделът на координация включва канали и механизми на взаимодействие.

Модел на данните
Моделът на данните се състои от водещите структури от данни и връзките между тях.
Важен елемент от модела на данните е начинът на съхранение в база от данни, във файлова система и т.н.

Управление на ресурсите
При управлението на ресурсите трябва да се определи кои ресурси и кои системни елементи ще се управляват; как тези ресурси ще си взаимодействат и какво въздействие ще имат върху процесора, изпълнението и т.н.

Връзки между архитектурните елементи
Връзките между архитектурните елементи могат да са между моделите, между елементите по време на изпълнение.
Такава връзка е разпределението на елементите по процесорите по време на изпълнение или пък на артикулите в модела на базата от данни.

Избор на технология
При избор на технологии трябва да се види кои технологии са налични.
Изборът на технология включва определянето на среда за разработка и среда за тестване.
Трябва да се отчитат и страничните ефекти от избора на технология.
Необходимо е да се следят зависимостите между технологиите и версиите им.

4.8. Обзор

Изискванията към системата се класифицират в три категории:

- функционални;
- атрибути на качеството;
- ограничения.

Атрибутите на качество се описват чрез сценарии със следните елементи: среда,

източник на стимул, стимул, артефакт, отговор, замерване на отговора.

5. Модул 5 Наличност (Availability)

5.1. Определение

Една система е важно да е в изправност; да може да извършва услугите, за които е предназначена; да бъде в готовност да отговаря на заявки; и бързо да се възстановява при провал.

При провал, системата изпитва затруднение да върши работата си или връща временно неверни резултати. Провалът може да прерасне в авария, ако системата не може да се възстанови от провала.

5.2. Замерване

Наличността на системата се замерва с времето на предоставяне на услугите, за които е предназначена т.е. времето за което е в изправност.

Availability = MTBF / (MTBF + MTTR)

MTBF – средно време на провалите

MTTR – средно време на отстраняване на провала

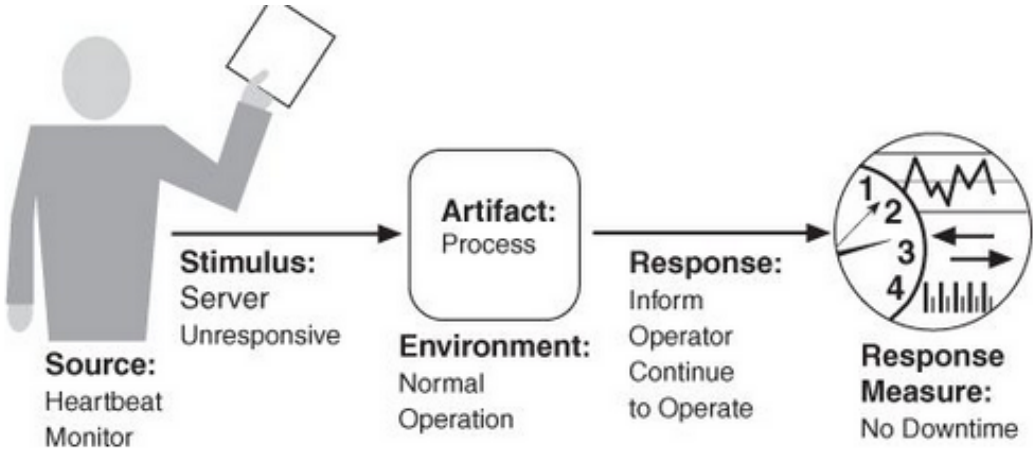
Брой деветки	Недостъпност за една година	Примерно приложение
3 деветки (99,9%)	~ 9 часа	Персонален компютър / Услуга
4 деветки (99,99%)	~ 1 час	Сървър на предприятие
5 деветки (99,999%)	~ 5 минути	Сървър за работа в реално време
6 деветки (99,9999%)	~ 31 секунди	Рутер за работа в реално време

Таблица 1. Класификация с деветките.

5.3. Проект на сценарий на наличност

- Източник на стимул – външен/вътрешен за системата
- Стимул – провал, срив: липса на отговор, счупване...
- Артефакт – хардуер, памет, процесор, канали

- Среда – нормална работа, временна натовареност
- Отговор - системата трябва да регистрира събитието; да извести, когато е необходимо; да забрани входа от източника; да остане недостъпна; да продължи работа в нормален или специален режим
- Замерване на отговора - време, за което трябва да е работоспособна; време на отстраняване на провалите; време, за което системата може да е с намалена работоспособност



Фиг. 17 Пример на сценарий – сървърът не отговаря.

5.4. Тактики за осигуряване на наличност

Фиг. 18 Цел на тактиките за наличност.

Фиг. 19 Тактики на наличност – йерархия.

Откриване на провали

Exo (Ping/Echo): компонентът А изпраща съобщение към компонента Б и очаква отговор в определен интервал от време.

Примерно използване:

- Клиент, проверяващ дали сървърът работи.
- Група компоненти, отговарящи за едни и същи задачи.

Пулс (Heartbeat - Keep Live): компонентът А периодично излъчва сигнал, очакван от компонента Б.

Сигналът може да излъчва и полезни данни, например:

- Банкоматът да изпраща данните от последната транзакция.
- Справка (log).

Изключения (Exceptions): обработват се изключения, които се генерират, когато се стигне до определено дефектно състояние.

Връзка между процедура на обработка и процес, генерирал изключението.

Гласуване (Voting) - едни и същи процесите се изпълняват на различни процесори; процесите имат един и същи вход и трябва да генерират еднакъв резултат; резултатите се гласуват за да се избере крайният резултат.

Ако някой от процесите произведе резултат, различен от останалите арбитърът решава да го изключи.

Отстраняване на провали

Гласуване (Voting) - едни и същи процесите се изпълняват на различни процесори; процесите имат един и същи вход и трябва да генерират еднакъв резултат; резултатите се гласуват за да се избере крайният резултат.

Ако някой от процесите произведе резултат, различен от останалите арбитърът решава да го изключи.

Активен излишък (Active redundancy, hot restart): важните компоненти се дублират; тези компонентите се поддържат в едно и също състояние приемат един и същи вход и извършват една и съща работа едновременно; използва се само един от еднаквите компоненти (активният).

Примерно използване: при клиент/сървър конфигурация, където се налага бърз отговор дори при срив.

Пасивен излишък (Passive redundancy): един от компонентите реагира на събитията; информира останалите компоненти; при дефект, се сменя активният компонент; синхронизацията се реализира от активният компонент – предава състоянието си през определен период от време.

Резерва (Spare): поддържат се резервни компоненти, които се включват при провал на основния компонент. Резервите не са активни.

Режим в сянка (Shadow mode): поправен модул работи паралелно в системата; не се въвежда в реална употреба, само се тества; ако работи коректно, се въвежда в реална употреба.

Синхронизация на състоянията (State Synchronization): прилага се в тактики изискващи въвеждането на резервни компоненти по време на работа - например пасивни и активни излишъци. Нужна е ***синхронизация на състоянието*** на компонентите преди да се вкарат в експлоатация. За предпочитане е синхронизацията да става с едно съобщение.

Връщане назад (rollback): прилага се когато системата трябва да се върне в предишно стабилно състояние. Текущото неизправно състояние може да е предизвикано от провал или дефект.

Софтуерно надграждане (Software upgrade): вкарва в употреба обновен стабилен код, премахва стария код с грешките му.

Предотвратяване на провали

Извеждане от работа (Remove from service): премахва компонент от работещата система с цел избягване на очаквани провали. Например, с рестартиране на компонент при установяване на изчерпване на паметта. Извеждането може да става ръчно или автоматично.

Транзакции (Transactions): няколко последователни стъпки, изменящи данни/състояние, се обединяват в една голяма. Осигуряват се атомарност, цялостност, изолация, и устойчивост на резултатите (ACID – atomic, consistency, isolation, durable) за данните.

Модел на предсказване (Predictive model): следи се за състоянието на системата и в

случай на очакван провал се предприемат действия за предотвратяването му. Например, ако много бързо се увеличава броя на потребителите на системата, се пуска нов екземпляр на компонента.

5.5. Контролен списък за проект на наличност

Разпределение на отговорностите – определена ли е функционалността, която трябва да бъде налична?

Модел на координация – гарантирано ли е, че всеки компонент или елемент данни могат да се координират в случай на провал? Координация трябва да се регистрира в журнал. Осигурява ли се смяна на артефактите при нужда.

Модел на данни: Кои елементи данни на системата са нужни наличност? Какви са операциите върху тях? Например: кеширане на данни.

Изобразяване между архитектурните елементи: Определени ли са кои артефакти (процеси, процесори, канали, хранилища) могат да предизвикат провал? Осигурени ли са тактики за връщането им в работен режим (рестартиране, заместване и т.н.)?

Управление на ресурсите: Кои са критичните ресурси, необходими за да поддържат системата работеща? Заделени ли са ресурси за възстановяване от провали?

Време на обвързване: Определено ли е кога и как архитектурните елементи се обвързват?

5.6. Обзор

Наличността се свързва с това, до колко една система е устойчива на провали и се намира в нормален режим на работа. Провалите могат да се разпознаят, предотвратят или поправят от системата.

Тактиките осигуряват наличност на системата.

6. Модул 6 Взаимодействие (Interoperability)

6.1. Определение

Как се интегрират готови услуги в новата система? Многократната употреба е от голямо значение, както за разработчика, така и за бизнеса – пести пари.
Как един е същи софтуер може да работи в различни системи?
Взаимодействието е свойството на различните системи да работят заедно.
Способността на две или повече системи или компоненти да обменят информация и да я използват е взаимодействие.

6.2. Аспекти на взаимодействието

Основните аспекти на взаимодействието са:

- комуникационните канали;
- транспорта на данни;
- съхраняването на данните;
- използването на стандартите;
- потребителският интерфейс;
- стека от технологии, използван в реализацията.

6.3. Проект на сценарий на взаимодействие

Източник на стимул: система инициира заявка към друга система.

Стимул: заявка за обмен на информация между системи.

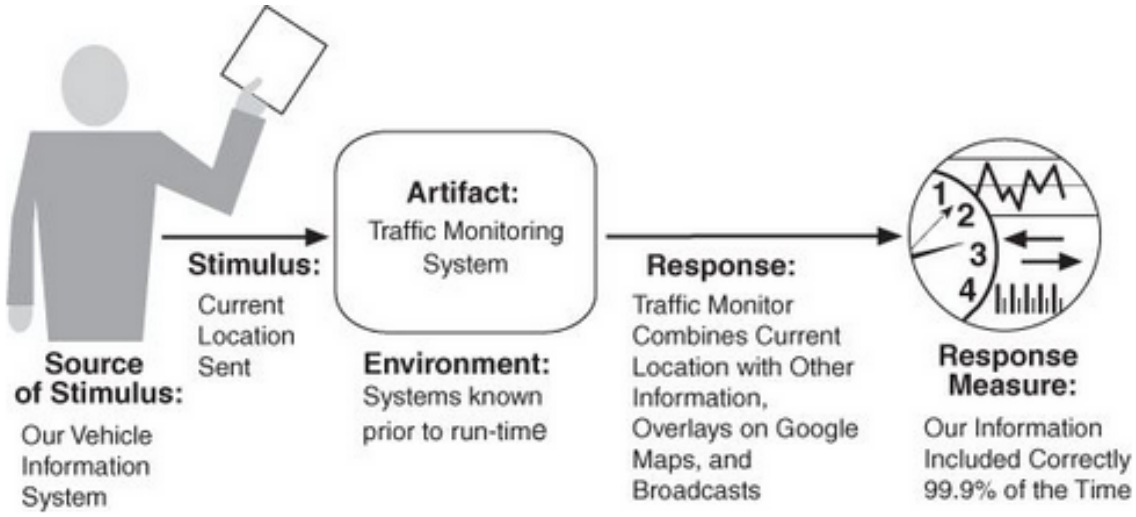
Артефакт: система, която иска информация от друга система.

Среда: Системи, взаимно използващи услугите си по време на изпълнение. Системите могат да се открият една друга и да се свържат.

Отговор: Заявката е отхвърлена и за това са уведомени съответните заинтересовани; или заявката е приета и информацията е успешно обработена; или заявката е регистрирана в една или повече системи.

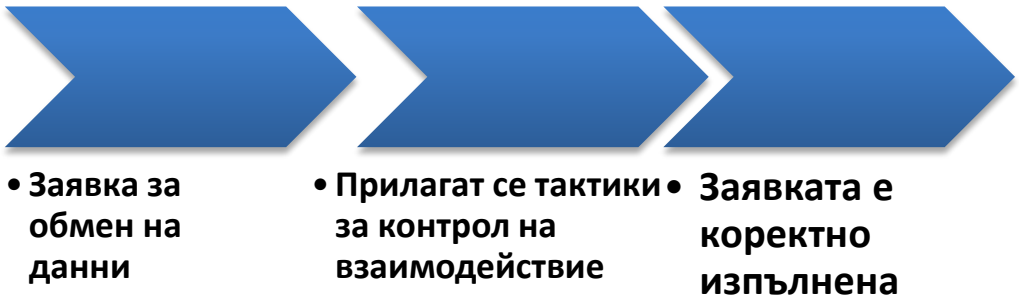
Замерване на отговора: процент на успешно изпълнените заявки между системи; процент на неуспешно изпълнени заявки между системите; брой на използваните външни системи.

Ф

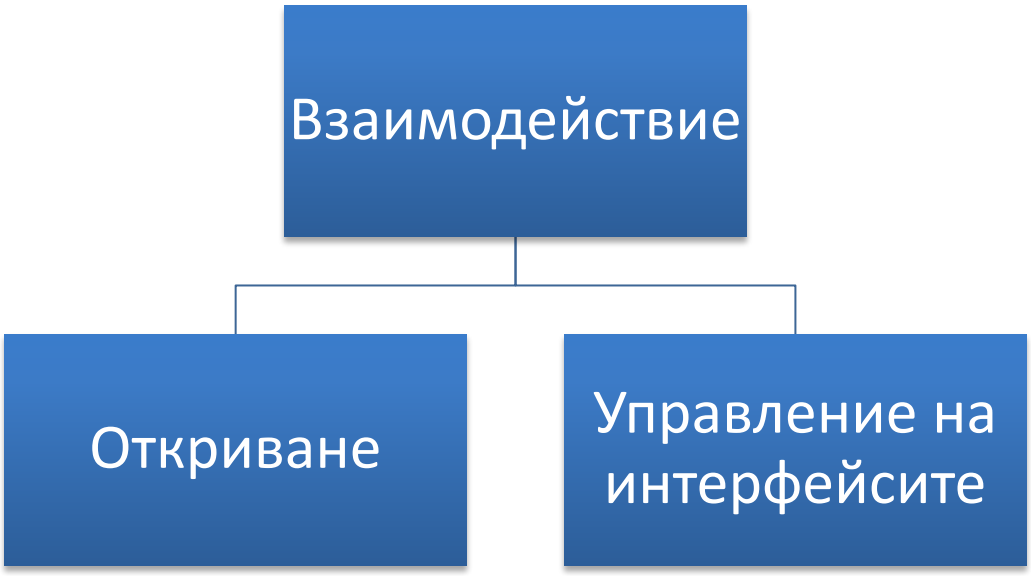


Фиг. 20 Сценарий на взаимодействие.

6.4. Тактики за осигуряване на взаимодействие



Фиг. 21 Цел на тактиките за осигуряване на взаимодействие.



Фиг. 22 Тактики за осигуряване на взаимодействие – основни групи.

Откриване

Откриване на услуга (Discover service): Системите, които ще взаимодействат по между си, трябва да могат да се открият една друга. Става дума за отдалечени услуги. Търсенето се извършва в специално предназначена директория. Критерии на търсене могат да са име, категория, местоположение и т.н.

Управление на интерфейсите

Оркестрация (Orchestrate): Специална система, контролира координация между услугите. Тя знае последователността на извикванията на услугите. Често се използва с образец медиатор.
Интерфейс по поръчка (tailor interface): добавя/изтрива функционалност към отдалечена услуга. Например: превод, буфериране на данни, премахване на дадени функции за неясни потребители. Системите на предприятието прилагат тази тактика с цел гъвкаво използване на ресурсите.

6.5. Контролен списък за проекта на взаимодействието

Разпределение на отговорностите: Кои елементи на системата имат нужда взаимодействие с други такива? Кои елементите, които ще работят с отдалечените услуги? Как ще се открива, свързва и приема/ отхвърля отговор на услуга от тези елементи? Кой и как ще регистрира извикването на услугата?

Модел на координация: Покрива ли механизма на координация изискванията за взаимодействие?

Относно изпълнението трябва да се съобрази трафика и поддръжката на състояние; необходимо време за изпращане на съобщението; необходимото време за сериализация/десериализация.

Относно наличността: дали са взети мерки в случай, че услугата е недостъпна.

Модел на данни: Дефиниран ли е модел/абстракция на данните при обмяна на заявките? Предвидени ли са преобразуватели на данните към модела на отдалечената услуга?

Изобразяване между архитектурните елементи: Критично е разположението на компонентите по сървърите и комуникацията между компонентите. Осигурени ли са наличност, сигурност и изпълнение?

Управление на ресурсите: Има ли увереност, че ресурсите се управляват според контракта на услугата?

Време на обвързване: Осигурена ли е политика на използване от други системи? Предвидено ли е отхвърлянето на времеотнемащи заявки? Осигурени ли са алтернативни услуги в случай на необходимост?

6.6. Обзор

Взаимодействието се отнася към възможността системата да обменя информация по подходящ за с другите системи.

Общият сценарий на взаимодействието е създаването и предоставянето на начини за обмен на информация.

Постигането на взаимодействието става чрез прилагане на тактики за откриване и за управление на интерфейсите.

7. Модул 7 Изменяемост (Modifiability)

7.1. Определение

Софтуерът търпи промени. Известна е максимата: “клиентите не знаят какво искат”. Бизнесът променя изискванията си поради промяна на пазара, което води до изчистване на целта на предлагания продукт/услуга.
Архитектурата трябва да е готова за тези случаи и да прилага промените, без това да струва твърде скъпо за бизнеса.
Изменяемостта на софтуера определя доколко лесно, ефективно и бързо може да се изменя една система.

7.2. Въпроси свързани с измененията

Въпросите са:

- Какво ще се променя? – функции, модул, капацитет
- Кога се случват промените? – в коя фаза на проекта – компилация, реализация, конфигурация
- От кого се правят промените? – от разработчика, системният администратор, самата система

- Каква е цената на промяната?

7.3. Измерване

Фиг. 23 Замерване.

Цената на въвеждане на механизъм улесняващ изменението

N – Сходни промени в изходен код

L – Цена на промяна без механизъм

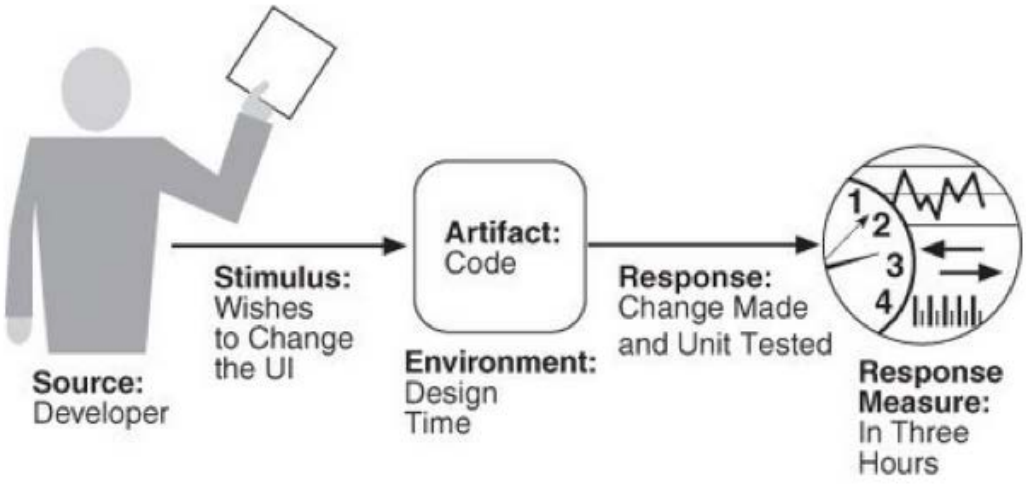
M – Цена на вграждане на механизъм

R – Цена на промяна с механизъм

$$N * L < M + (N * R)$$

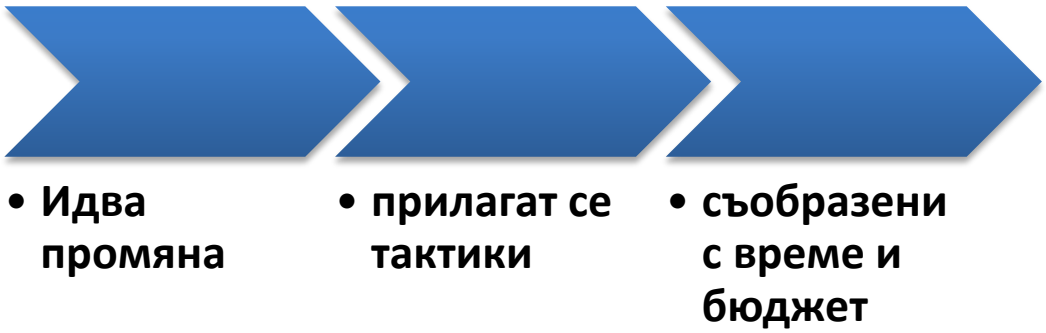
7.4. Проект на сценарий за изменяемост

Източник на стимул: разработчици, системни администратори, крайни потребители, системата.
Стимул: добавяне, премахване, промяна на функционалност, капацитет и т.н.
Артефакт: модул от системата, потребителският интерфейс, изходен код, цялата системата и т.н.
Среда: по време на разработка, по време на изпълнение, компилация, конфигурация и т.н.
Отговор: намират се местата подлежащи на промяна, правят се промените без това да се отрази на останалите елементи/останалата функционалност.
Измерване на отговора: време, цена, усилия – стойност като цяло.



Фиг. 24 Сценарии на изменяемостта.

7.5. Тактики за осигуряване на изменяемостта



Фиг. 25 Цел на тактики за осигуряване на изменяемостта.

Водещи параметри при прилагането на тактиките

Висока кохезия (high cohesion) – модулът има една единствена отговорност.
Слаба свързаност (low coupling) – до колко промяната в един модул влияе върху друг? Налага ли се промяна в друг модул?
Размер на модул (module size) – големина в изходен код на модул.

Фиг. 26. Тактики за осигуряване на изменяемостта – основни групи.

Висока кохезия

Целта е да се намали броя на модулите, в които се правят изменения. Така се намалява цената на изменението.
Идеята е обхватът на измененията да е в минимален брой модули.
Семантична съгласуваност (semantic coherence): При разпределението на отговорностите в даден модул неговите задачи и отговорностите да не зависят много от другите модули. Постига се чрез събиране на цялата семантично свързана функционалност в отделен модул.
Намаляване на размера на модула (Reduce size of module): Модул с прекалено отговорности е твърде тежък за поддържане и за това се разделя на подмодули – по възможност по отговорности и по задача за реализация.
Очаквани изменения (Changes expectation): Прави се списък на вероятните изменения. **Подпомагащи въпроси са:**

- Помага ли направеното разделение на модули по лесни евентуалните промени?
- Случва ли се така, че функционално различните изменения да засягат един и същи модул?

Изследва се ефекта от евентуални изменения. Тази тактика е трудна за осъществяване!
Помощни модули (Utility): Това са модули, класове, в които има обща функционалност за другите модули. Примерно: математически операции, работа с дати, файлови операции и други.

Намаляване на свързаността между модулите

Още се казва: “Намаляване ефекта на вълната”.
Идеята е да се намалят измененията в модулите, които не са пряко засегнати от

дадена промяна.
Пример: Модул А зависи от модул Б, ако промяната в модул А зависи от изменението в модул Б.
Тази категория тактики, намалява зависимости между модулите.
Капсулация (Encapsulation): Във всеки модул, част от информацията е публична, а другата се скрива. Публичната информация се вижда от другите модули (чрез API, публичен метод или интерфейс). Скрива се информация, която няма отношение към външните модули. Пряко е свързана с тактиката на очакваните изменения: списъкът на очакваните промени е водещ при декомпозицията.
Интерфейс/приложен програмен интерфейс (Interface/API): Модул А да зависи от модул Б само по неговия интерфейс. При необходимост се добавя нов интерфейс, нова обвивка (wrapper, adapter), и т.н. за да не се изменя оригиналния интерфейс.
Ограничаване на комуникацията (Communication constraint): ограничава се броя на модулите, с които даден модул комуникира; ограничава се информацията, която се предава; ограничават се модулите, които създават информация.
Използване на посредник (Use an intermediary): Ако модул А зависи от модул Б по някакъв начин, е възможно да се добави посредник, който премахва тази зависимост.
Идентификация на интерфейси (contract identification): Ако модул А предоставя няколко интерфейса , а модул Б има нужда само от един от тези интерфейси, тогава този конкретен интерфейс може да стане достъпен чрез брокер. Модул Б се свързва с брокера и той го свързва с този интерфейс.
Съществуване на А (A existence): за да може А да се изпълни е необходимо Б да съществува. Например, А изисква услуги от Б. Ако Б липсва, може негов екземпляр да се създаде динамично. Това може да стане чрез метод на фабрика или чрез образеца на абстрактната фабрика.
Услуги/качество на данните (services/data quality): Има две основни изисквания за коректно прилагане на тези тактики:

- ***Семантика на данните:*** модул А предава данни във вид очакван от модул Б;
- ***Семантика на услугите***: модул А предоставя в семантична обвивка очакваните услуги от модул Б и последният знае как да ги използва.
  - Важна е и последователността:
- ***За данните***: модул А подава данните на модул Б в определена последователност;
- ***За изпълнението***: модул А изпълнява услугата в определена времева рамка, за да може модул Б да се работи коректно.

Отлагане на свързването

До тук разгледаните тактики засягат програмирането, но има тактики, които не зависят от програмирането.
Може да се инвестира в допълнителна инфраструктура, с което да се намали зависимостта между модулите.
Включи и ползвай (Plug-and-play): включване, изключване, замяна на компоненти. Може по време на изпълнение или по време на зареждане да се избира компонент.
Конфигуриране на файлове (File configuration): задават се стойности на

различните параметри; стойностите зависят от интерфейса или реализацията на използвания модул. Например: връзката към базата от данни да е според сървъра – тестова или продукционна.

7.6. Контролен списък за проверка на изменяемостта

Разпределение на отговорностите: Кои отговорности или функционалности на системата са засегнати? Кои елементи са засегнати?
Координационен модел: Кои функционалности или комуникационни канали могат да се променят по време на изпълнение? Кои канали и инструментите могат да се използват за промени?
Модел на данни: Кои данни ще се променят или добавят? Какъв вид промяна се налага? Кои данни имат влияние върху промените?
Изобразяване между архитектурните елементи: Кои са зависимите артефакти по време на изпълнение? Кои са процесите, нишките, процесори, засегнати от промените?
Управление на ресурсите: Как добавянето, променянето или изтриването на функционалност или отговорност засяга използването на ресурсите? Кои промени налагат добавянето на нови ресурси или отстраняването им? Кои граници на ресурсите се променят и как?
Време на обвързване: Кога промяната трябва да се осъществи? Каква е цената и времето за изпълнение? Кои са възможните пречки?

Обзор

Изменяемостта се свързва с промяната и стойността й се измерва във време или пари. Тя има влияние върху модулите и/или функциите на системата. Измененията се правят от софтуерните инженери, потребителите или системата.
Тактиките намаляват цената на изменението.

8. Модул 8 Изпълнение (Performance)

8.1. Определение

Някои въпроси свързани с изпълнението:

- Колко е важно за една система да си свърши работата на време?
- Кой и как дефинира това време?
- Колко е важно за една уеб базирана система да извършва 100 транзакции за 1 минута?
- Колко време е необходимо на софтуера на кола да запали двигателя или да реагира при натискане на спирачката?

Предмет на изпълнението е времето, за което системата реагира на възникващи събития, предизвикани от потребители или външни системи.

Едно архитектурно решение има добра производителност, когато потребителите изпълняват задачите си чрез системата в поставената времева рамка и това не влияе на останалите атрибути за качество.

8.2. Термини свързани с изпълнението

Време за възстановяване (Recovery time) е времето, необходимо за възстановяване от отказ/провал.

Време за отговор (Response time) е времето, необходимо на обработка на една заявка.

Време за спиране (Shutdown time) е времето, необходимо на системата да спре в нормални условия.

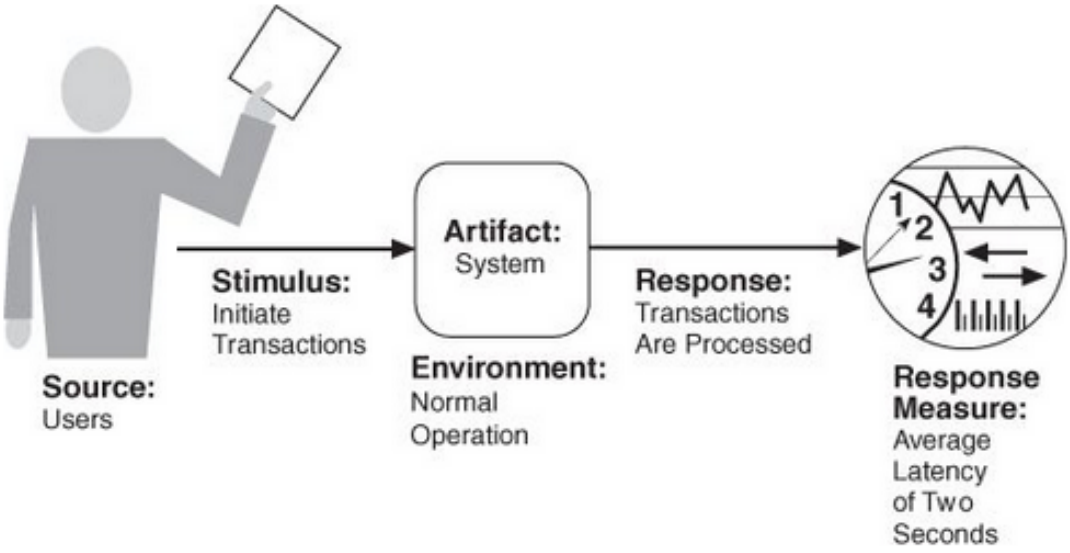
Време за стартиране (Startup time) е времето, необходимо на системата да стартира в нормални условия.

Пропускателна способност (Throughput) е количество работа, извършено от системата за фиксиран период от време.

Забавяне (Latency) е времето между началото на заявката и отговорът на системата на тази заявка.

8.3. Проект на сценарий за изпълнението

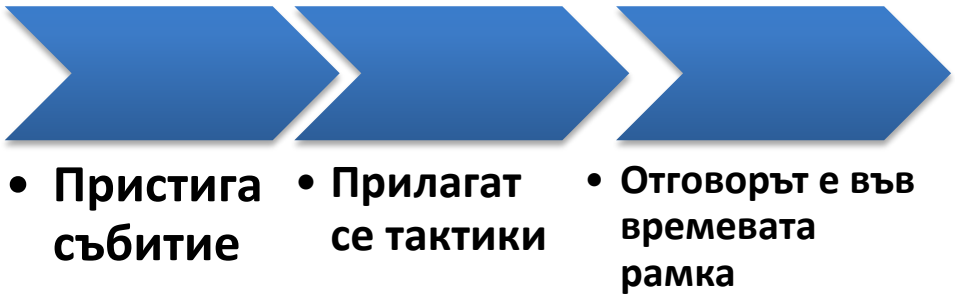
Източник на стимул: множество от източници, потребителски заявки, други системи, вътрешни за системата източници.
Стимул: може да е периодично, стохастично или спорадично събитие.
Артефакт: системата или конкретен процес в нея.
Среда: нормален режим, режим на претоварване.
Отговор: обработва събитието и/или променя качеството на обслужването.
Замерване на отговора: латентност; навременност; пропускливост; отклонение; брой необработени/обработени заявки.



Фиг. 27 Сценарий на изпълнение.

8.4. Тактики за осигуряване на изпълнението

За да реагира системата е нужно време, защото са необходими ресурсите за обработката, а също така има заключване на ресурсите при конкурентно изпълняващите се процеси.



Фиг. 28 Цел на тактиките за осигуряване на изпълнението.

Целта е ясно дефинирана за подобен вид качествени атрибути – имайки предвид производителност в случая, тя ще е времето необходимо за обработка на заявката да е ограничените в поставените й за това времеви граници
Водещи параметри при прилагането на тактиките:

- Време за обработка (processing time)
- Време за изчакване (block time)

- Политика на използването на ресурсите - няколко елемента искат да използват един и същи ресурс.

Фиг. 29 Тактики за осигуряване на изпълнението – основни групи.

Ограничаване на нуждата от ресурси

Ограничаване на времето за отговор на събитие (limit event response): Някои събития се поставят в опашка. За тези събития не е важно да се отговори веднага. Те се обработват когато има свободни ресурси.
Увеличаване на периода на изпълнение на събитието (Increase period event execution): Прилага се при периодични събития. Чрез увеличаване на периода се намаляват изискванията към ресурсите. Пример: изпращане на електронна поща с уведомление. Тогава се събират статистики за потребителя и системата. Периодът за събиране на статистиката може да бъде увеличен от един път дневно на един път седмично.
Увеличаване ефективността на ресурса (Increase resource efficiency): Идеята е в намаляване времето, необходимо за извършване на определена функция, например, чрез подобряване на алгоритмите или чрез кеширане - за по-бърз достъп до данните.
Ограничаване на времето за отговор на събитие (limit event response): Някои събития се поставят в опашка. За тези събития не е важно да се отговори веднага. Те се обработват когато има свободни ресурси.
Увеличаване на периода на изпълнение на събитието (Increase period event execution): Прилага се при периодични събития. Чрез увеличаване на периода се намаляват изискванията към ресурсите. Пример: изпращане на електронна поща с уведомление. Тогава се събират статистики за потребителя и системата. Периодът за събиране на статистиката може да бъде увеличен от един път дневно на един път седмично.
Увеличаване ефективността на ресурса (Increase resource efficiency): Идеята е в намаляване времето, необходимо за извършване на определена функция, например, чрез подобряване на алгоритмите или чрез кеширане - за по-бърз достъп до данните.
Намаляване на предварителната обработка (Reduce overhead): събитието се обработва във верига от компоненти с различни отговорности, а не в монолитен

компонент. Може да не се извършва обработка, която няма връзка със събитието. Може да се осигурят се медиатори в обработката.

Управление на ресурсите

Конкурентно изпълнение (concurrency execution): Заявките обработват конкурентно. Процесът не изчаква завършване на предходните операции. Пример: процесът се представя чрез паралелни нишки.
Увеличаване на ресурсите (Increase resources): Добавят се повече процесори, памет, хранилища, мрежа. Обикновено, струва пари, но значително намалява забавянето. Трябва да се търси се златната среда между покупка на ресурси и изпълнение.
Режимиране (Scheduling): При спор за ресурси, диспечерът взима решението. Събитията могат да имат приоритети – по-важните са с предимство пред диспечера. Важно характеристика е предаване на управлението на ресурса между събитията. Алгоритми, които се прилагат за обслужване: стек, фиксиран приоритет, дек, динамичен приоритет.

8.5. Контролен списък за проект на изпълнението

Разпределение на отговорностите: Коя функционалност на системата се поддържа от процеси консумиращи много ресурси? Кой реализира контрола на нишките.
Модел на координация: Кои елементи на системата трябва да се координират – пряко или непряко? Необходими ли са конкурентни процеси? Какви са свойствата на комуникационните механизми: без състояние или състояние, синхронни или асинхронни.
Модел на данни: Кои данни са важни и кои отнемат много време за обработка? Разделянето на данни, дали няма да е от полза? Дали тактики, като увеличаване на изчислителни ресурси, може подпомогнат операциите с тях – създаване, модифициране и т.н.
Изобразяване между архитектурните елементи: В кои части на системата може да има голямо натоварване на мрежата? Всички процеси с интензивни изчисления дали се изпълняват върху мощни процесори?
Управление на ресурсите: Кои ресурси са критични за изпълнението? Използвани ли са процеси, нишки и приоритизирани ли са?
Време на обвързване: Зададено ли е време за изпълнение?

8.6. Обзор

Изпълнението се отнася за начина управление на ресурсите в зададена времева рамка.
Подобряването на изпълнението зависи от прилагането адекватни за това тактики.

9. Модул 9 Сигурност (Security)

9.1. Определение

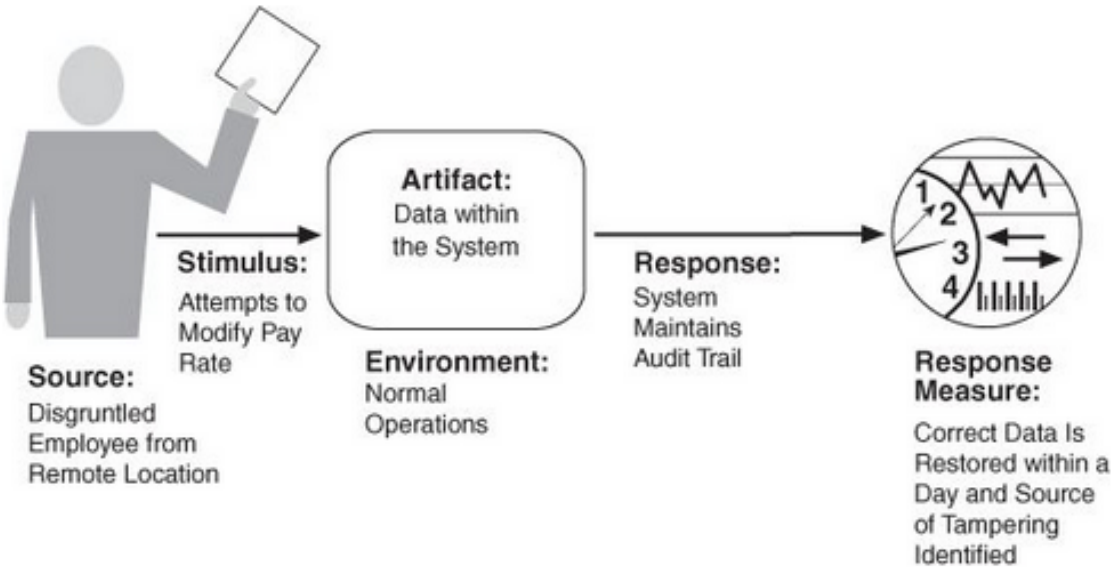
Система без сигурност, е като пускане на домашно животно в дивата природа.
Как в онлайн система за продажби се осигуряват сигурни разплащания?
До каква степен данни на потребителите са защитени?
Кой потребител ще има вяра в система без сигурност?
Сигурността е мярка за способността на системата да устоява на опитите за неразрешена употреба, без това да пречи на легитимните потребители.

9.2. Речник в сигурността

Конфиденциалност (**Confidentiality**): свойство на услугите да не се изпълняват от лица, които не притежават разрешение за това.
Цялостност (**Integrity**): свойство на данните да се доставят във вида, в който са предадени.
Наличност (**Availability**): свойство на системата да бъде в наличност за легитимно използване.
Удостоверяване (**Authentication**): проверка на потребителите/елементите, че са тези, за които се представят.
Безотказност (**non repudiation**): гаранция, че изпращачът не може да отрича, че е изпратил съобщението и на получателя, че го е получил.
Разрешаване на достъп (**Authorization**): дава на потребителя права за достъп до данни и услуги.
Атака (**Attack**): опит да се компрометира сигурността: кражба на пари, кражба на номера на кредитни карти, атака с цел отказ от услуга и т.н.

9.3. Проект на сценарий за сигурност

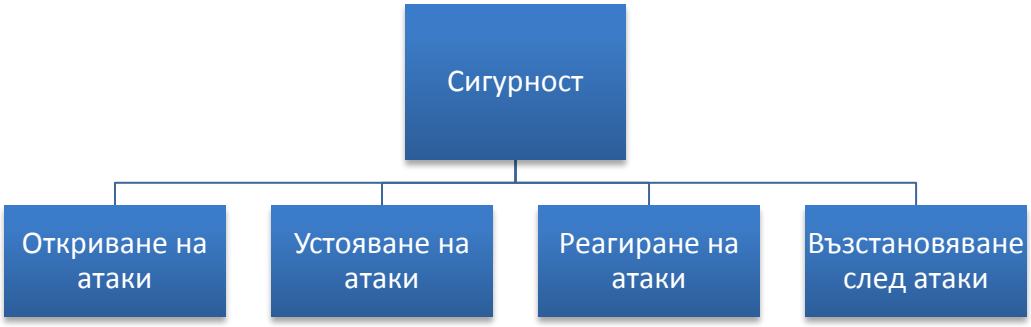
Източник на стимул: човек или система, които са правилно/неправилно идентифицирани; могат да са външни или вътрешни; може да имат права за достъп или на нямат такива.
Стимул: опит за разкриване на данни; опит за промяна или изтриване на данни.
Артефакт: услуги на системата или данни.
Среда: на линия или изключена; свързана или несвързана; пред или зад защитна стена.
Отговор: допуска легитимния потребител и скрива идентичността му; блокира достъпа до данни за нелегитимните потребители; записва опитите и т.н.
Замерване на отговора: време, усилия, ресурси; процент на работоспособност на системата; обхват на пораженията.



Фиг. 30 Сценарий на сигурност.

9.4. Тактики за осигуряване на сигурността

Фиг. 31 Цел на тактиките за осигуряване на сигурността.



Фиг. 32 Тактики за осигуряване на сигурност – основни групи.

Откриване на атаки

Система за откриване на нахлуване (IDS – Instruction Detection System): Системата анализира трафика, сравнява с база от данни за познати атаки.

Устояване на атаки

Удостоверяване на потребителя (User authentication): проверява се дали потребителят е този, за когото се представя. Използват се пароли, имена на потребители, сертификати и т.н.
Идентифициране на актьори (Identify actors): Идентифицира се източника на връзката със системата. Например, потребителите се идентифицират чрез потребителските им имена, а външните системи чрез IP адресите им.
Разрешаване на достъп на актьорите (Authorize actors): Идентифицираният потребител има права да изменя данните или ползва услугите на системата. Контролът на достъп до системата се реализира чрез задаване на роли на потребители или групи.
Ограничаване на достъпа (limit access): ограничава се достъпа до компютърни ресурси, памет, мрежови комуникации и т.н. Постига се чрез защитна стена, блокиране на сървъри, отхвърляне на протокол и т.н.
Криптиране на данни (Encrypt data): Прилага се за защита на данни от неразрешен достъп. Поверителността се постига чрез криптиране на данните в комуникационният канал. Примери за такъв вид защитени интернет връзки за VPN и SSL.

Възстановяване след атаки

Свързани са с възстановяване на състоянието на системата след атака.

Извършителите се идентифицират и се проследяват извършените от тях транзакция със системата.

Реагиране на атака

Заключване на компютър (Lock computer): Неколкократните опити за влизане в системата предизвиква заключване на компютъра или компонента. Обикновено, това се прави за определено време.

9.5. Контролен списък за проект на сигурността

Разпределение на отговорностите: Кои отговорности/функционалности се нуждаят от сигурност? За всяка от тях трябва да има:

- Идентифициране на потребителите;
- Удостоверяване на потребителите;
- Разрешаване на достъп на потребителите;
- Допускане или блокиране на достъпа до данните и/или услугите;
- Способност за възстановяване от атака;
- Верификация с проверочни суми или хеш стойности.

Модел на координация: Има ли механизъм за вътрешна комуникация, която поддържа криптиране, удостоверяване, разрешаване на достъпа, наблюдение и записване на действията?

Модел на данни: Кои са чувствителните данни? За различните видове данни има ли различни права на достъп? Криптират ли се данните и кога? Възстановяват ли се данните след неразрешени изменения?
Изобразяване между архитектурните елементи: Какви са връзките между архитектурните елементи в контекста на сигурността? За актьори предвидени ли са идентификация, удостоверяване, разрешаване на достъпа? Предвидено ли е възстановяване след атака?
Управление на ресурсите: Кои ресурси ще се наблюдават? Контролира ли се външния/вътрешния достъп и неговото ниво?
Време на обвързване: Има ли механизъм за валидиране на данните? Колко е време за достъп до услугите за сигурност?

9.6. Обзор

Атаките в системата могат да се класифицират като атаки към конфиденциалността, атаки към цялостността на данните и атаки към достъпа до системата като цяло. Тактиките като идентифициране, удостоверяване, криптиране, следене и т.н., предпазват системата от тези видове атаки.

10. Модул 10 Възможности за тестване (Testability)

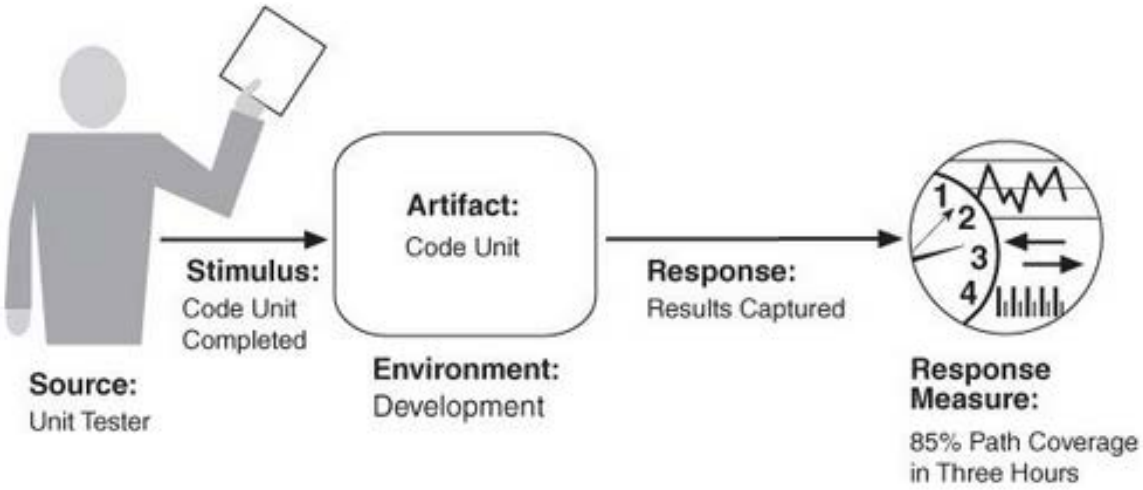
10.1. Определение

40-50% от времето за разработка на софтуер минава в тестване. Струва си архитектът да се опита да намали тази стойност. Тестването включва наблюдение, както на входни, така и на изходни параметри.
Възможностите за тестване се отнасят към лекотата, с която софтуерът може да бъде накаран да покаже дефектите си посредством изпълнението на различни тестове.

Фиг. 33 Видове тестове.

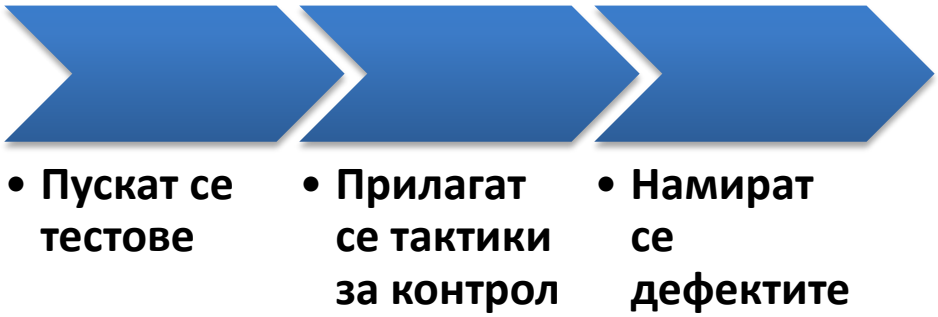
10.2. Проект на сценарий на тестване

Източник на стимул: разработчик, интегратор, специалист по тестовете, клиента.
Стимул: завършена е версия на архитектурата; завършена е интеграция на подсистемите; системата е цялостно завършена.
Артефакт: части от анализа, проекта, парчета код, или цялата система.
Среда: по време на разработването, по време на проектирането, по време на компилация и внедряване.
Отговор: дава се достъп до вътрешните променливи и състоянието, до изчислените стойности. Това става чрез тестова установка.
Замерване на отговора: процент на изпълнените операции; процент покритите потребителски случаи на системата; вероятност за регистриране на дефекти.



Фиг. 34 Сценарии за тестване

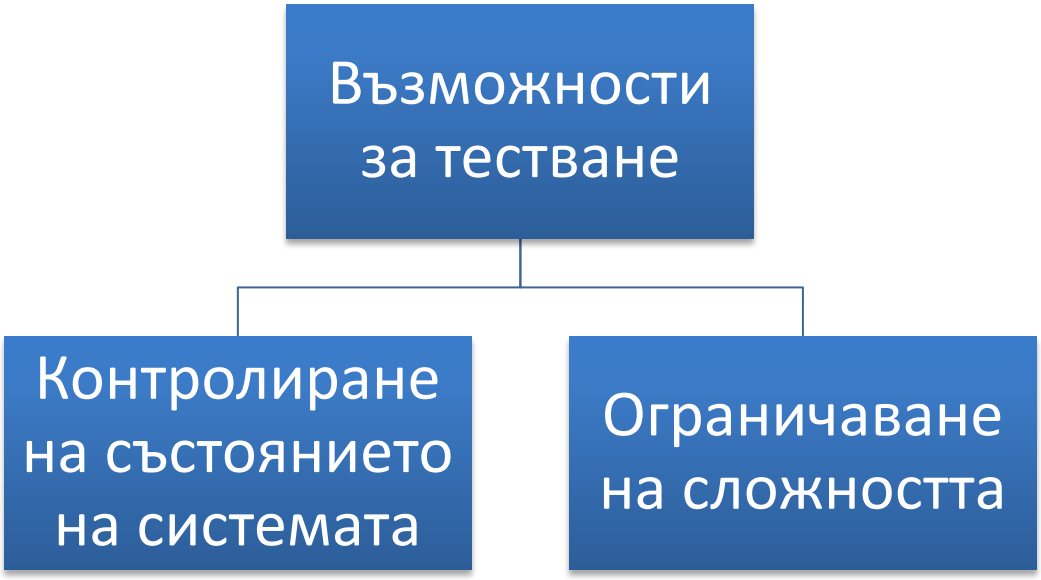
10.3. Тактики за осигуряване на тестването



Фиг. 35 Цел на тактиките за осигуряване на тестването.

Тактики за осигуряване на тестването - инструменти

Тактиките подпомагат тестването и процеса чрез преглед на кода или чрез система за изпълнение на тестването. Последната приема входни параметри и анализира резултата; работи на принципа на „черната кутия“ – само с интерфейсите.



Фиг. 36 Тактики за осигуряване на тестването – основни групи

Контролиране на състоянието на системата

Специализация на интерфейсите (Specialized interfaces): отделяне на контрактите от реализацията. Спомага да се замерва на коректността на реализацията.
Запис/възпроизвеждане (Record/Playback): Използва се прихващане на входната информация и използването ѝ в следващи тестове. Тази информация може да се подаде на тестовата установка за софтуера или да се генерират на входните данни. Изходните състояния се записват за последващи сравнения.
Наблюдение (Monitoring): наблюдава се информация за състоянието, натовареността, изпълнението, сигурността. Събраните данни се използват за тестване на системата.
Интерфейси за изпълнение на тестове (Tests harness interfaces): Те са различни от нормалния интерфейс. Позволяват управлението на тестването чрез метаданни. Сравняват се резултатите с тези от нормалния интерфейс.

Ограничаване на сложността

Ограничаване на структурната сложност (Limit structural complexity): осъществява се чрез поддържане на висока кохезия (high cohesion), слаба свързаност между компонентите (low coupling), избягване на цикличните зависимости и ограничаване на височината на йерархиите от класове.

10.4. Контролен списък за проект на тестване

Разпределение на отговорностите: Кои елементи са най-критични и изискват тестване? Осигурено ли е тестването им и анализ на резултатите? Записва ли се информацията от тестванията?

Модел на координация: Осигурено ли е тестването на елементите? Предвидени ли са дейности при отказ на системата? Има ли наблюдение на системата?
Модел на данни: Кои абстрактни данни ще се тестват? Могат ли да се чете и разбира входа и изхода от тези данни? Операциите върху данните тестват ли се?
Изображение между архитектурните елементи: Как се тестват връзките между архитектурните елементи – нишки към процеси, модели към компоненти и т.н.
Управление на ресурсите: Осигурени ли са ресурси за тестване и анализ на резултатите? Предвиден ли е анализ на неуспехите при тестването?

11. Модул 11 Използваемост (Usability)

11.1. Определение

Колко бързо потребителят извършва задачата си използвайки системата?
Каква помощ му се оказва?
Достатъчно интуитивен ли е един сайт?
Използваемостта се отнася към това, колко **лесно** потребителят да **върши** дадена задача и каква **подкрепа** му се оказва за това.

11.2. За какво се използва?

За запознаване с характеристиките на системата и нейните свойства.

За ефективно използване на системата.

За минимизиране на последствията от грешките на потребителя.

За адаптиране към нуждите на потребителя.

За увеличаване на увереността на потребителя в ползването на системата.

11.3. Метрики за използваемостта

Продуктивност:

- ***време*** за извършване на конкретна задача;
- ***процент*** на грешките и въздействие.

Разбиране:

- ***използване*** на познати образци на работа;
- ***разпознаване на стойността.***

11.4. Проект на сценарий на използваемостта

Източник на стимула: крайният потребител.

Стимул: Потребителят иска да използва ефективно системата, да намали ефекта от грешки си, да се чувства уверен, да адаптира системата.

Артефакт: системата или конкретна част от нея.

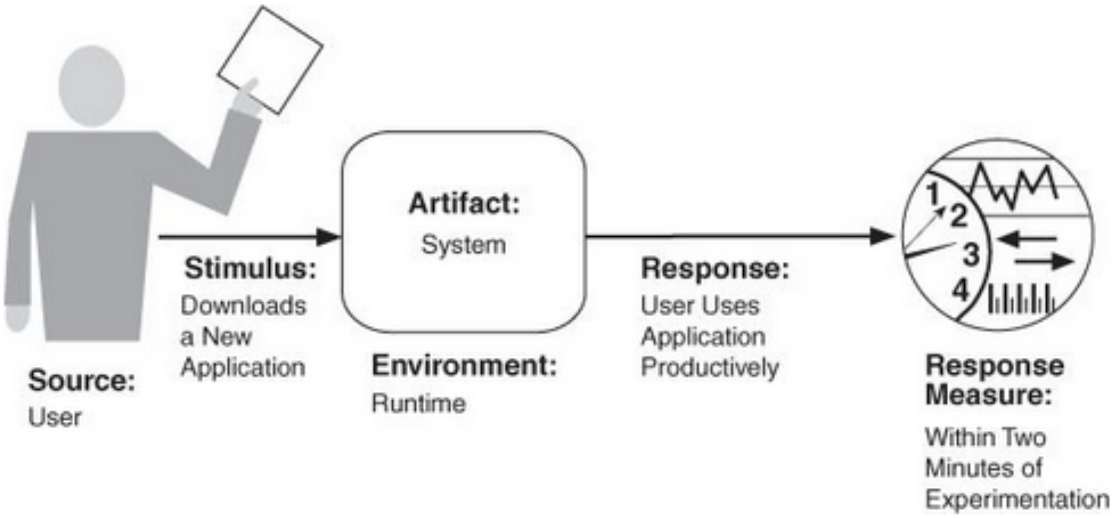
Среда: по време на изпълнение или по време на конфигуриране.

Отговор:

- **За научаване на нова функционалност:**
  - стандартизиран интерфейс, познат на потребителя;
  - Контекстно-ориентирана система.
- **За постигане на ефективност:**
  - агрегация на функции;
  - ефективна навигация;
  - постоянство в интерфейса;
  - разширено търсене.
- **За намаляване ефекта от грешки:** отмяна, отказ, възстановяване от грешка, възстановяване и отстраняване на грешка потребителя.
- **За постигане на увереност:**
  - показване на моментното състояние;
  - показване на прогреса при дългите операции;
  - визуална обратна връзка;
  - работа в ритъма на потребителя.

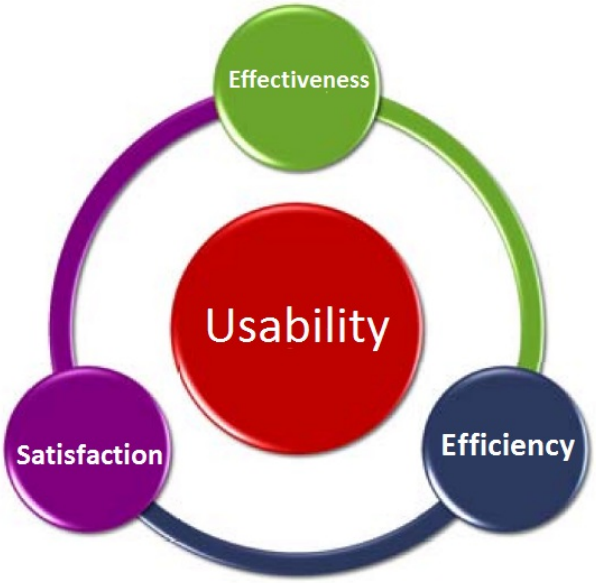
Замерване на отговора:

- време за извършване на дадени задачи;
- брой грешки;
- брой на решените проблеми;
- удовлетвореност на потребителя;
- получаване на нови знания;
- отношение на успешните към неуспешните операции;
- изгубено време;
- изгубени данни.

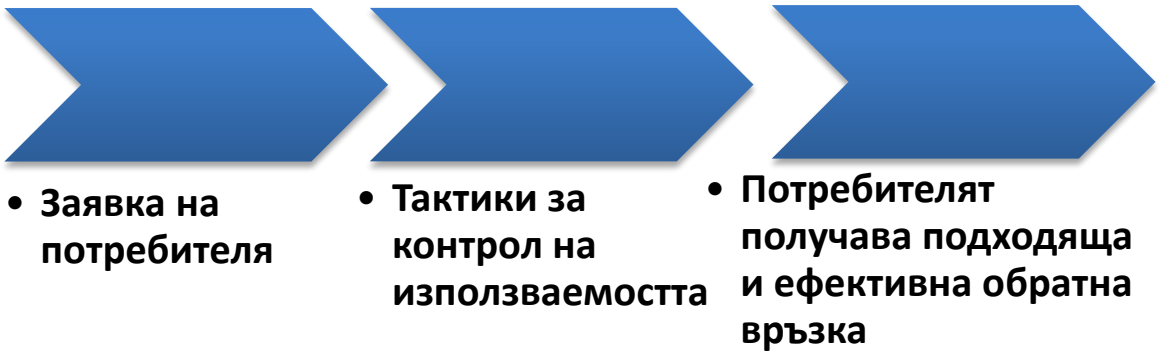


Фиг. 37 Сценарий на използваемост.

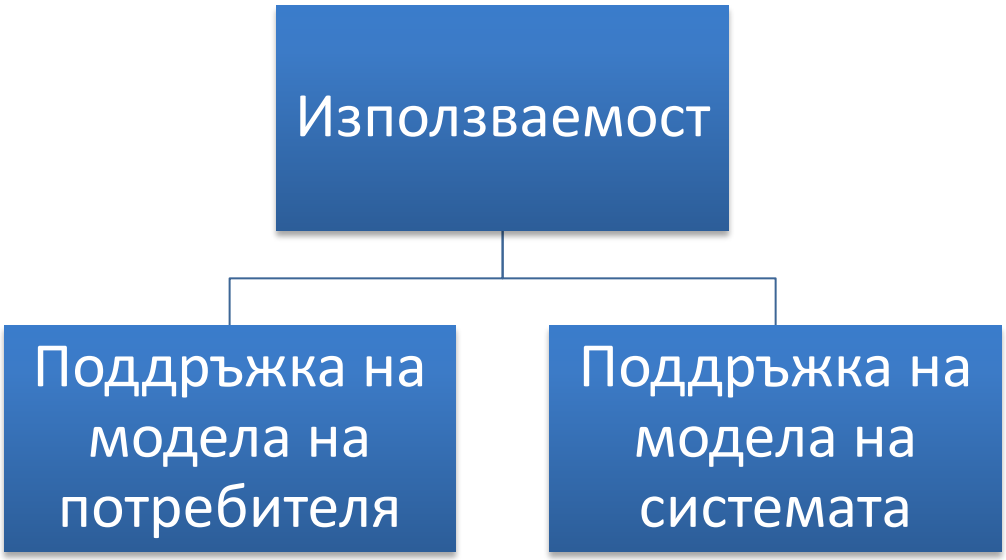
11.5. Тактики за осигуряване на използваемостта



Фиг. 38 Тактики за осигуряване на използваемостта.



Фиг. 39 Цел на тактиките за осигуряване на използваемостта.



Фиг. 40 Тактики за осигуряване на използваемостта – основни групи

Поддръжка на модела на потребителя

Отказ (Cancel): Системата „слуша" за отказ на заявка. Използваните ресурси се освобождават.
Отмяна (Undo): Предишните състояния се пазят и при тази заявка системата се връща към предишно състояние. Няма ефекти върху другите части на системата.
Пауза/подновяване(Pause/resume): Това е възможност дългите изпълнения да бъдат временно спрени и по-късно подновени от момента на тяхното прекъсване.

Поддръжка на модела на системата

Поддържане на модел за задачите (Maintain task model): В определен контекст, системата разпознава какво върши потребителят и му помага/подсказва.
Поддържане на модел на потребителя (Maintain user model): Моделът представя потребител/група от потребители и позволява персонализиране на интерфейса.
Тактики за използваемост – примери:

- Бутоните за навигация използват различни цветове, когато са селектирани или активни.
- Извикването на действия имат ясен език и стил на представяне.
- Голям информационен сайт има функция за търсене по текст.

11.6. Контролен списък за проект на използваемостта

Разпределение на отговорностите: Има ли помощ за потребителя в обучението за използване на системата? Има ли възстановяване от грешки на потребителя/системата?
Модел на координация: Има ли координация между системните елементи при изучаване от потребителя на системата и при изпълнение на задачите със системата.
Модел на данни: Създадени ли са моделите според нуждите на потребителите? Налични ли са структури и операции, покриващи задачите на системата? Пример:

поддържане на операции за отмяна и отказ.
Изобразяване между архитектурните елементи: Ясни ли са елементите за потребителя? Например, когато потребителят използва отдалечени услуги. Наясно ли потребителя с ефекта при използването на елементи на трети страни?
Управление на ресурсите: Как потребителят задава и използва конфигурации на системата? Има ли помощ за потребителя при конфигуриране?
Време на обвързване: Кои части от системата кога трябва да се контролират, имайки в предвид времето им на ползване от потребителя?

11.7. Обзор

Архитектурата спомага за използваемостта. Тя дава възможност на потребителя да поема инициативи – например, да прекрати дълго изпълняващи се операции или да се откаже от последната команда.
Системата трябва да може да придвижва реакцията на потребителя.
Системата трябва да дава възможност на потребителя да бъде ефективен в работата си чрез прилагане на тактики.

12. Модул 12 Други атрибути за качество

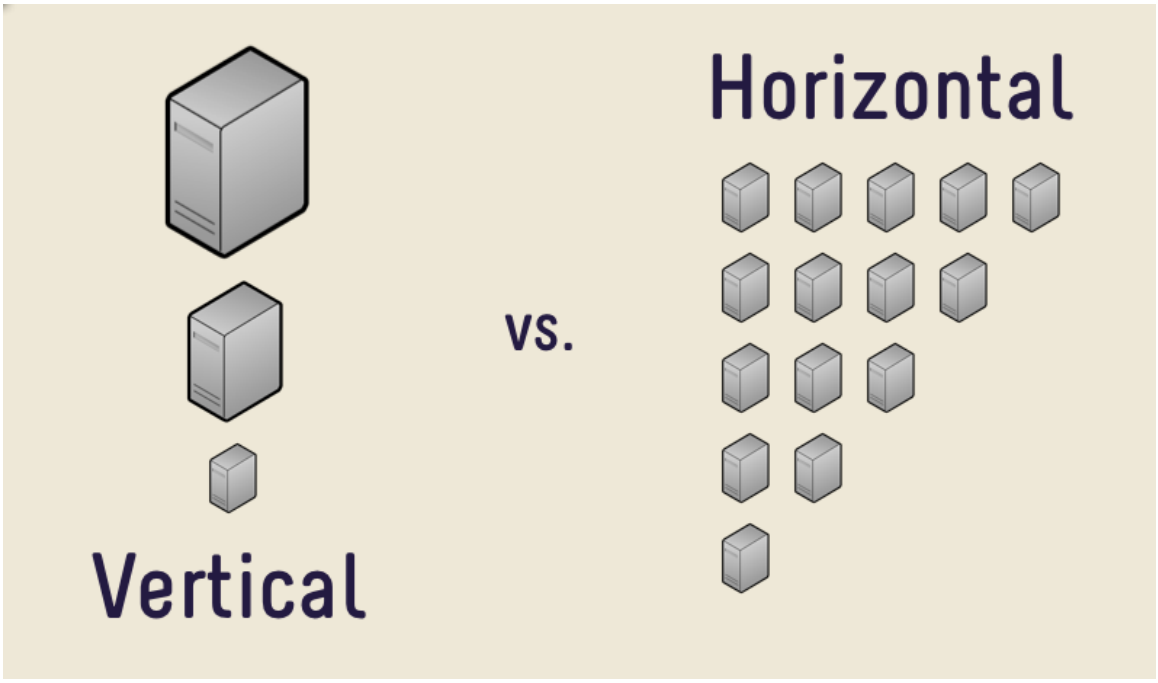
12.1. Увод

До тук описаните атрибути за качеството са важни за софтуерният продукт или услуга.
Всеки атрибут за качество се дефинира чрез сценария си за качество и за всеки от тях има тактики за постигането им. Така се дефинират атрибутите за качество.
От атрибутите за качество се избират и приоритизират тези, които са необходими за постигане целите на системата.
Постигането на атрибутите за качество се измерва с метрики.

12.2. Други важни атрибути за качество

Променливост (*Variability*): Това е специална форма на изменяемостта. Тя дава възможност системата и/или нейните артефакти, лесно да се конфигурират и да са адаптивни към различни потребителски случаи. Този атрибут се използва най-вече в продуктовите линии.
Преносимост (Portability): Това е специфична форма на изменяемостта. Тя отговаря на въпросите: Как софтуерът да се пакетира и разгръща върху различни платформи? Колко бързо това може да стане?
Постига се чрез: минимизиране на зависимостите в системата; изолиране на зависимостите в строго определени места; софтуер, изпълняван на виртуални машини (Java Virtual Machine).
Мащабируемост (*Scalability*): Определя колко лесно може една система да промени броя на потребителите си. Мащабирането може да е нагоре или надолу, но и в двата случая става дума за работа в нормален режим.
Разгръщаемост (*Deployability*): Отнася се за разгръщането на системата върху нова платформа, върху облак и т.н. От значение е времето, да което това ще стане. Разглеждат се сценарии на разгръщане.
Мобилност (*Mobility*): Отнася се за пренос системата към различни мобилни устройства. Важни характеристики са пропускателна способност, честота на приемане, живот на батерията. Отговаря на въпроси от вида на:

- Как се подържат различните видове потребителски интерфейси?
- Как се управлява батерията?



Фиг. 41 Хоризонтална и вертикална мащабируемост.

Наблюдаемост (*Monitorability*): Свойство на системата да бъде наблюдавана и следена за важните й характеристики като:

- дължина на опашките от съобщения;
- средно време за обработка на транзакция;
- надеждност на различните компоненти на системата.

12.3. Други категории атрибути за качество

Бизнес качества

Това е набор от нефункционални бизнес изисквания. Те се формират се по подобен на технологичните качества. Примери:

- себестойност;
- време за изработка;
- пазарни условия.

Време за излизане на пазара (Time to market): Определя се от пазарния натиск. Рефлектира върху технологичните характеристики за качество на софтуера. Обикновено, времето се намалява с използването на готови компоненти.

Себестойност и печалба: Определя се от съотношенията: проект и бюджет; архитектура и технологии. Непознатите технологии могат да струват скъпо. Наблюдава се зависимост между технологичните и бизнес атрибутите за качество. Пример: Високо изменяемата система струва повече, пести в поддръжка.
Време на живот: Отчита се в набора от технологични атрибути на качеството и зависи от планираното време за експлоатация на системата. Например, ако системата ще се експлоатира дълго, то тя трябва да е мащабируема, изменяема и преносима.

Архитектурни качества

Идейна цялост: Основополагаща концепция и визия за проекта на всички нива. Софтуерната архитектура се прави, така че подобните задачи да се решават с подобни решения.
Коректност и пълнота: Дали софтуерната архитектура позволява спазване на изискванията и ограниченията?
Примери:

- Себестойност – метрика най-вече в стойност на пари
- Време на изработка
- Пазарни условия - какъв е маркетинга в момента на разработване на идеята ? Кои са конкурентите ? Има ли такива ? Каква е добавената стойност на продукта/услугата ?

12.4. Атрибути на качеството на софтуера и на системата

Има връзка между софтуерните и технологичните атрибути на качеството. Интензивните на изчисления системи се нуждаят от повече хардуерна мощ, памет и бързи процесори, а това струва пари.
Трябва да се търси баланс между атрибутите.
Софтуерът е ограничен от системата, върху която ще се изпълнява:

- Бързодействието на софтуера зависи от бързодействието на компютъра, върху който се изпълнява.
- Мащабируемостта зависи от компютрите, сървъра и техните свойства за увеличаване на памет, създаване на нови екземпляри и т.н.

12.5. Стандартните списъци за атрибути на качеството

Има стандарти, дефиниращи списък от софтуерни атрибути на качеството. Такъв е ISO/IEC FCD 25010.
В него атрибутите се разпределят „качество на използването“ и „качество на продукта“.
ISO 25010 - списък за качеството на продукта:

- Функционална пригодност (Functional suitability)
  - коректност на изискванията;
  - компетентност;
  - завършеност.
- Ефективност на изпълнението (performance efficiency):
  - бързодействие;
  - капацитет.
- Съвместимост (Compatibility):
  - Оперативна съвместимост
- Използваемост (Usability):
  - достъпна система за потребителя;
  - обратна връзка;
  - лекота на научаване;
  - завършеност на задачите;

  - защита от потребителските грешки.
- Надеждност (Reliability)
  - цялостност;
  - наличност;
  - възстановяване при провал.
- Сигурност (Security):
  - конфиденциалност на данните;
  - удостоверяване;
  - цялостност.
- Поддържаност (Maintainability):
  - модулност;
  - многократно използване;
  - лекота на тестване;
  - анализируемост на данни и процеси.
- Преносимост (Portability):
  - адаптивност;
  - инсталируемост;
  - копия и репликации.

12.6. Справяне с „X-способност”

Често един атрибут на качество зависи от други атрибути, рефлектира върху свойствата на системата, оказва влияние върху другите качества (положително и отрицателно), рефлектира върху метриките на другите атрибути.
Нов атрибут за качество трябва да се въведе в последователни стъпки, обхващайки цялата картина – архитектура, софтуер, сценарии.

1. Създаване на сценарии за новия атрибут за качество:
   - Интервю със заинтересованите лица относно новия атрибут на качество.
   - Създаване на различни потребителски случаи за новия атрибут на качество.
   - Създаване на различни сценарии.
   - Обобщаване на сценарий по общия образец.
2. Сглобяване на проектни решения за постигане на новия атрибут за качество:
   - Разглеждане на готови проектни образци.
   - Търсене на проект, който се справя с подобни атрибути за качество.
   - Намиране на експерти в областта на разглеждания атрибут за качество.
   - По сценария се търсят подходи.
   - По сценария се търсят подводни камъни и начини (грешки, неправилни отговори) на справяне с тях.
3. Сглобяване на тактики за постигане на атрибута за качество: Има два основни типа тактики моделни и експертни. При моделния процес на генериране на тактики се търсят и изброяват параметрите на модела, а за всеки параметър се избират архитектурни решения имащи влияние върху него.

13. Модул 13 Архитектурни тактики и образци

13.1. Архитектурни образци - основни постановки

Проектирането на софтуер е трудно, тъй като е свързано с много проблеми. Постигането на многократна употреба е още по-трудно, понеже трябва да се открият подходящите обекти, да се разпределят по класове, да се дефинират интерфейсите им, да се организират в йерархии на наследяване, да се фиксират връзките между тях. Обикновено, проектът е специфичен за текущия проблем, но трябва да е достатъчно общ, за да може да посрещне бъдещи промени и проблеми.
“Преоткриването на топлата вода” не е необходимо всеки път да се прави всеки път. Могат да се използват предишни решения, които са доказано и добри. Добрият проектант е този, който използва готовите, доказаните решения.
Образците решават специфични „проектантски“ проблеми, а не специфичен проблем.
Образците правят системата по-гъвкава, по-елегантна. Те са с многократна употреба.
„Наистина новите“ проекти отнемат много време и ресурси, за да се докаже и провери, че решават добре проблема, докато старите вече са се доказали.
Образците документират опита при архитектурното проектиране. Те са успешни решения, но не са панацея – не решават всеки проблем.
Образците предлагат алтернативи, а алтернативата е свободата на избор между две или повече решения. Тогава може да бъде избрано по-доброто.
Образците помагат при документирането на проекта.
Много е трудно разбирането на програмите, които ги използват, ако не се знаят образците им.

13.2. Архитектурни образци

Архитектурните образци са:

- **Решение**, което описва обекти, класове и взаимоотношенията между тях.
- **Решаван проблем**: те решават общ проектен проблем.
- **Контекст**: приложими са в даден контекст.

Връзката *{решение – проблем - контекст}* документира архитектурния образец.

Архитектурните образци идентифицират участващите класове и обекти, техните роли, взаимоотношения и отговорности.

Архитектурните образци предлагат примерен код.

Някои езици за програмиране и платформи съдържат реализирани части от образците.

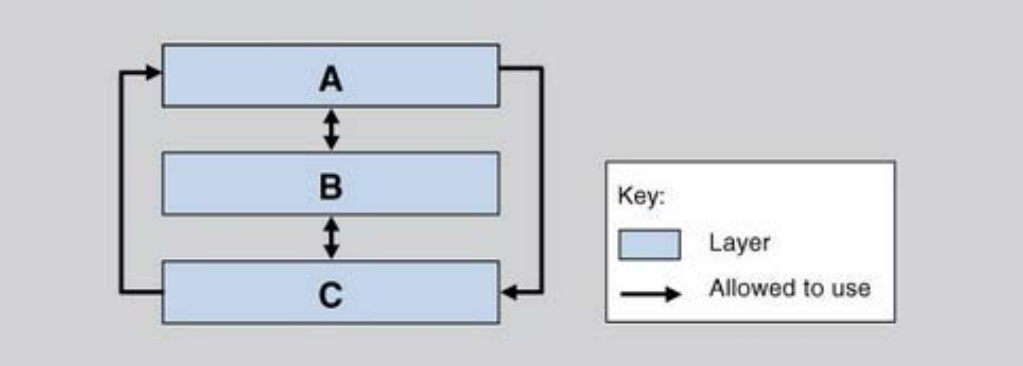
13.3. Преглед на каталога от образци

Основни елементи на образците са решавания проблем, решението, контекста на решението и допълнително може да включва връзки, елементи, ограничения.

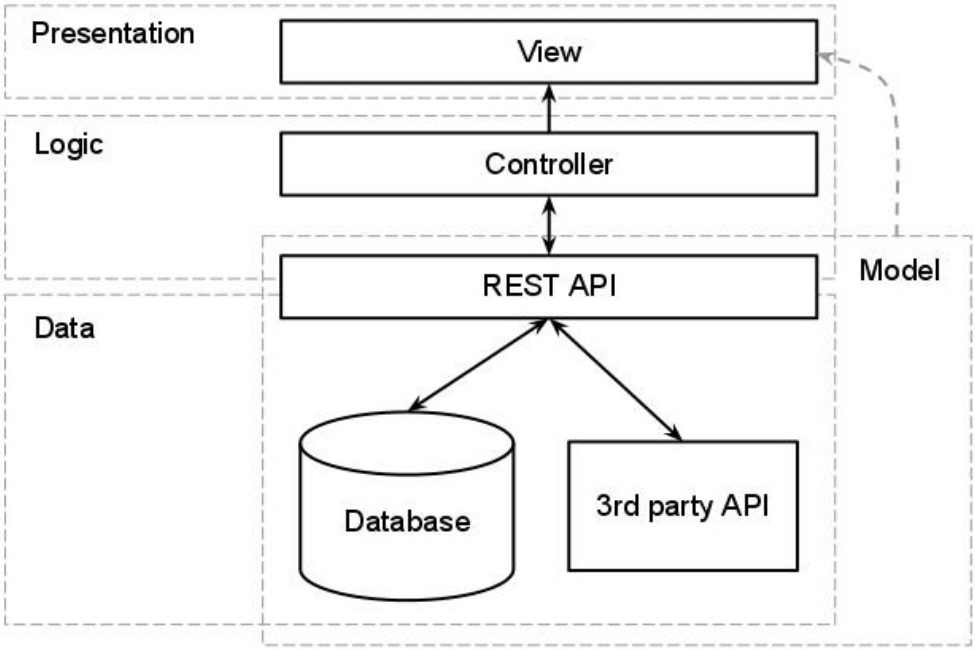
Образците се разделят по приложни области.

Образец на слоевете (Layered pattern)

Контекст: Сложните системи изискват независима разработка на отделните части от софтуера. Същността е да се разделят отговорностите по приложната логика.
Проблем: Елементите трябва да се разделят, така че да се разработват поотделно. Връзките между тях трябва да са минимални. Трябва да се поддържат преносимост, изменяемост и многократна употреба.
Решение: Софтуера се разделя на слоеве. Всеки слой е силно свързан. Всеки слой използва по-долния слой – връзката е в еднопосочна.
Ограничения: Всяка част от софтуера е в един единствен слой. Има най-малкото два слоя. Начинът на използване не трябва да позволява циклични зависимости.
Слабости: Всеки нов слой прави системата сложна. Многото слоеве могат да забавят изпълнението.



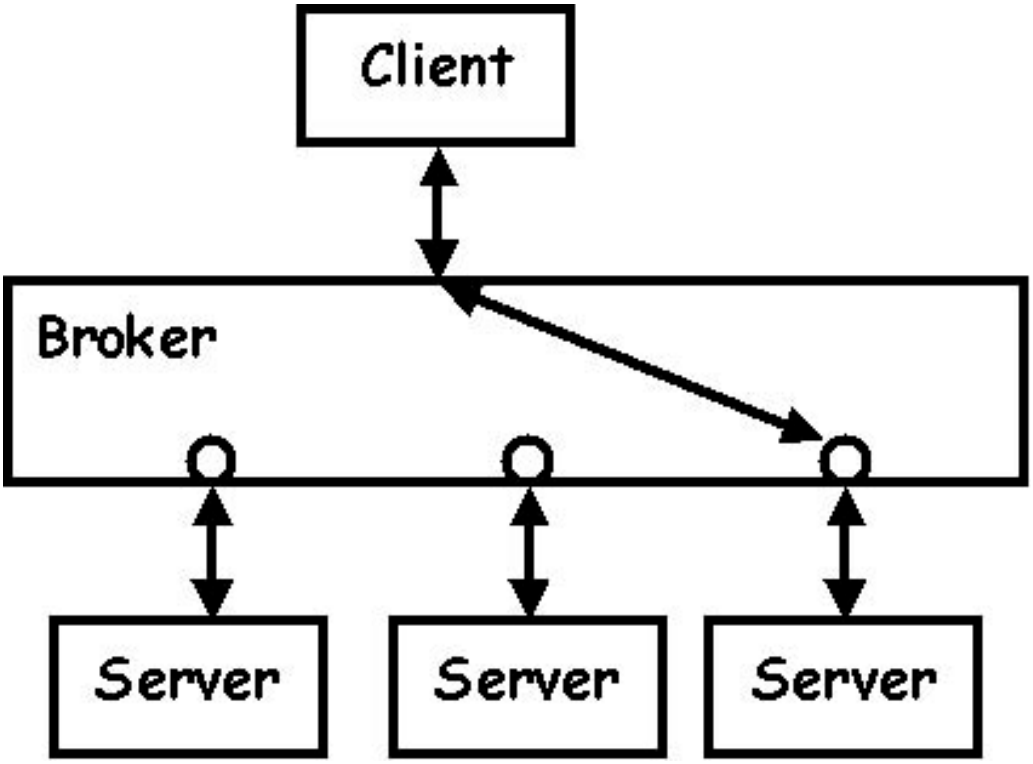
Фиг. 42 Нотация на образеца на слоевете.



Фиг. 43 Пример на образец на слоевете.

Образец с брокер (Broker pattern)

Контекст: Система е конструирана като набор от услуги, разпределени върху множество сървъри. Необходима е връзка между услугите.
Проблем: Как услугите се свързват по между си? Как научават местоположението на нужните им услуги? Как обменят данни услугите?
Решение: Образецът разделя потребителите на услугите от доставчиците на услуги. Това става с добавянето на брокер (посредник), който предава заявките между клиент и сървър.
Брокерът получава заявката от потребител Х и я праща на услуга У, намираща се на сървър Д.
Прокси (proxy) често използва този образец.
Ограничения: Клиентът може да се свързва само с брокера. Сървърът може да се свързва само с брокера.
Слабости: Има по-голямо забавяне между клиенти и сървъри заради брокера. Последният би могъл да бъде единствената точка на отказ.



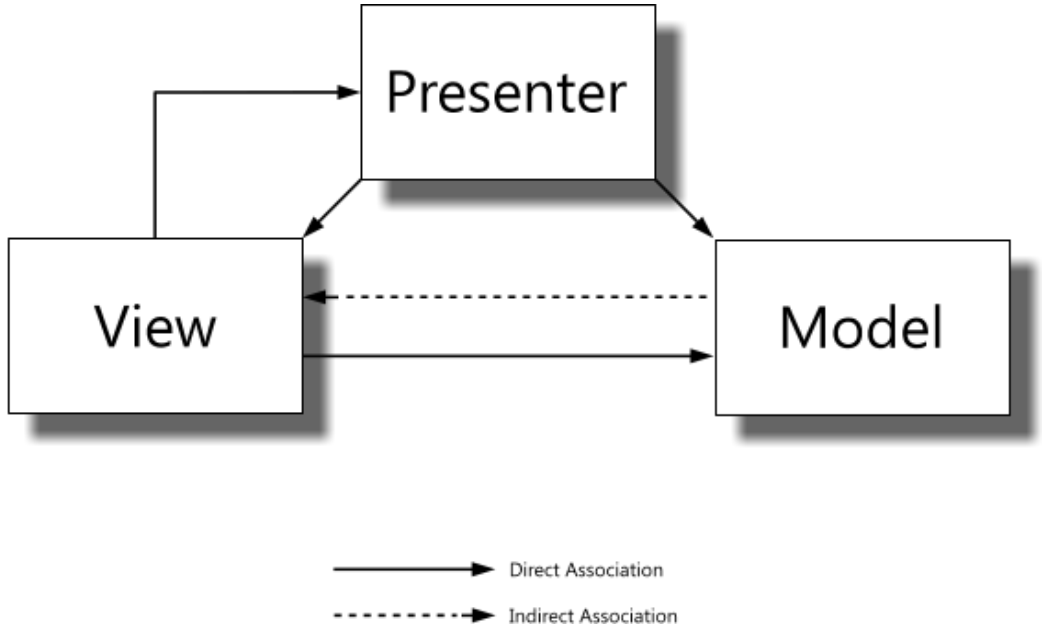
Фиг. 44 Пример с брокер.

Модел-гледка-контролер(Model View Controller - MVC)

Контекст: Потребителският интерфейс е най-често променящият се софтуер при интерактивните системи. За това е важно да се отделят промените върху интерфейса, така че да не влияят върху останалата част от системата. Потребителите често искат да гледат на системата от различни гледни точки (различни гледки).
Проблем: Как функционалността на системата може да се разграничи от потребителският интерфейс и в същото време да отговаря на потребителските заявки в различен контекст? Как различните гледки на интерфейса да се създават, изменят и премахват?
Решение: MVC разделя системата на 3 основни компонента:

- модел: съдържащ данните на приложението;
- гледка: визуализираща данните;
- контролер: посредник между данните и гледките.

Ограничения: Нужни са поне по един екземпляр на модела, гледката или контролера. Моделът не трябва да взаимодейства пряко с контролера.
Моделът капсулира състоянието на приложението. Той Отговаря на заявките за състоянието и откроява функционалността на приложението. Уведомява гледките за промените.
Гледката визуализира моделите, изисква обновяване от моделите и дава възможност на контролерите да избират гледките.

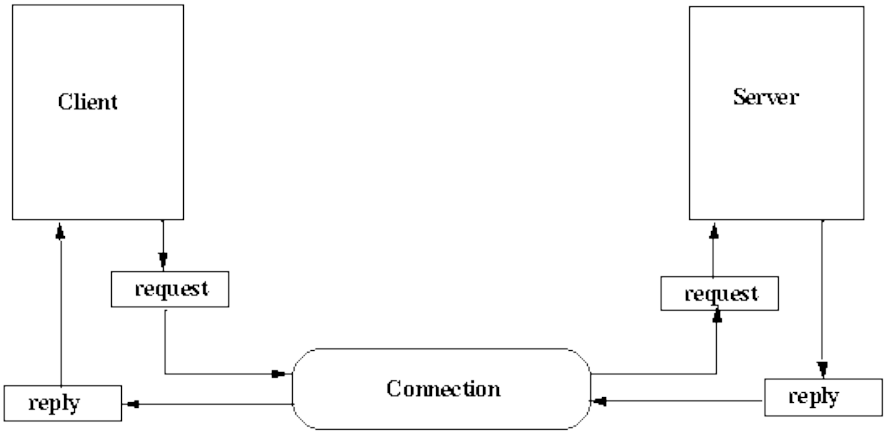


Фиг. 45 Пример на модел-гледка-контролер.

Контролерът определя поведението на приложението, свързва потребителска заявка с модела, избира гледка за отговора. Той е единствен за всяка функционалност.

Клиент-сървър (Client-Server)

Контекст: Налични са различни ресурси и услуги и множество от потребители искат да имат достъп до тях. Достъпът трябва да е контролиран.
Проблем: Трябва да има една точка за контрол на достъпа. Да има възможност за мащабиране.
Решение: Клиентите взаимодействат със сървърите за да ползват техните услуги.
Ограничения: В броя на връзките към един порт – сървър.
Слабости:
Сървърът може да е тясно място в изпълнението. Сървърът може да е единствената точка на провал.



Фиг. 46 Пример с клиент-сървър.

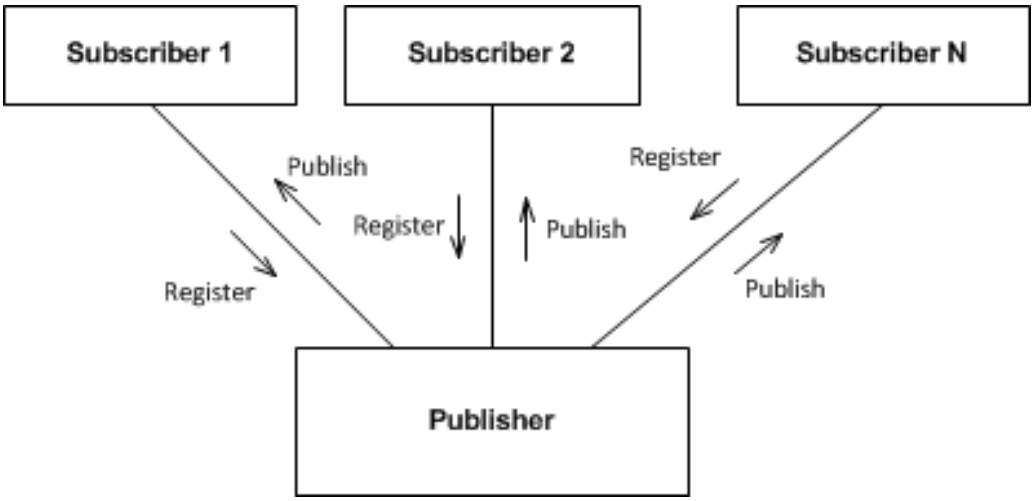
Ориентирана към услуги архитектура (SOA)

Контекст: Набор от услуги се предоставя от доставчици на услуги. Тези услуги се консумират от различни потребители или системи. Консуматорите на услуги трябва да познават синтаксиса и начина на използването на услугите.
Проблем: Как може да се реализира взаимодействие между различните услуги, реализирани на различни платформи и езици за програмиране? Как може да се откриват и комбинира услугите?
Решение: SOA описва набор от разпределени компоненти, които предоставят и/или консумират услуги. Доставчикът на услугата и консуматорът на услугата могат да са разработени с различни технологии и езици за програмиране. Услугите обикновено се изпълняват независимо една от друга. Атрибутите на качеството на услугите е зададено в споразумението за ниво на услугата (SLA – Service Level Agreement).
Ограничения: Използва се медиатор за връзка между консуматорите и потребителите на услуги.
Слабости: Обикновено, услугите са сложни за реализация. Услугите трудно се дефинират. Медиаторите имат отрицателно влияние върху изпълнението.
Инфраструктурата на медиаторите между услугите е шината на услугите в предприятието (ESB – Enterprise Service Bus). Тя маршрутизира съобщенията между консуматорите и доставчиците на услуги, сериализира и десериализира съобщенията.
Слоят на оркестрация оркестрира изпълнението на услугите. Оркестрацията описва последователността от извиквания на услугите.
Конекторите се реализирани с протоколи SOAP, REST, асинхронни съобщения и други.

Публикуване/Абониране (Publisher/Subscriber)

Контекст: Има набор от независими консуматори и производители на данни. Те

трябва да се интегрират помежду си.
Проблем: Как може да се създаде механизъм, който да подпомага размяната на съобщения между производителите и консуматорите на данни?
Решение: Компонентите си взаимодействат чрез съобщения или събития. Абонатите се абонират за събития. Когато има събитие всички абонирали се, получават съобщението на това събитие. Всеки компонент може да публикува и да е абонат.
Ограничения: Всеки компонент трябва да се свърже с дистрибутор на събития (шина). Ограничава се кои компоненти какво слушат (кои събития).
Слабости: Шината може да е тясно място на изпълнението. Няма гаранция за получаване на съобщенията.



Фиг. 47 Пример с публикуване/абонамент.

13.4. Връзка между тактики и образци

Образците и тактиките са двигателят при избора на технологии и софтуерна архитектура.
Образците са комбинация от тактики.
Тактиките са „атомите”, образците са „молекулите”.
Тактиките се използват за постигане на определени атрибути на качеството.
Тактиките могат да се използват за подобряване на атрибутите на качеството на образците.

	Modifiability									
	Increase Cohesion		Reduce Coupling					Defer Binding Time		
Pattern	Increase Semantic Coherence	Abstract Common Services	Encapsulate	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Startup-Time Binding	Use Runtime Binding
Layered	X	X	X		X	X	X			
Pipes and Filters	X		X		X	X			X	
Blackboard	X	X			X	X	X	X		X
Broker	X	X	X		X	X	X	X		
Model View Controller	X		X			X				X
Presentation Abstraction Control	X		X			X	X			
Microkernel	X	X	X		X	X				
Reflection	X		X							

Фиг. 48 Връзка между тактики и образци.

Тактики за подобрения на образеца брокер

Брокерът има проблеми със следните атрибути на качеството:

- Наличност - брокерът е единствена точка на провал.
- Изпълнение - забавяне на заявките, тъй като минават задължително през брокера.
- Трудно се тества поради много процесори и процеси.
- Сигурността – брокерът може да е основен обект на атака.

Подобрения чрез тактики:

- Изпълнението може да се подобри, ако се използват няколко брокери.
- Балансиране на изпълнението може да се постигне чрез синхронизация между брокерите.
- Чрез тактиките за проверка на активност, потребителят може да разбере кога един брокер е аварирал.

Обзор

Един архитектурен образец е набор от архитектурни решения, които се използват често в практиката. Образецът имат свойства, които позволяват многократно да се използва.
Образецът описва клас от архитектури.
Архитектурен образец се дефинира с:

- Контекст: повтаряща се ситуация, в която има проблем.

- Проблем: обобщен за даден контекст
- Решение: успешно архитектурно решение на проблема на достатъчно абстрактно ниво.

14. Модул 14 Моделиране и анализ на атрибутите за качество

14.1. Увод

Анализът на софтуерната архитектура допринася за предвиждане на атрибутите й за качество.
Без анализа не е ясно дали са избрани подходящите структури от елементи, дали те реализират функционалността на системата, дали се постигат важните свойства и стойности на метриките на системата.
Анализът спомага да се следи постигането на желаните стойности на метриките.

14.2. Моделиране на архитектурите с цел анализ на атрибутите за качество

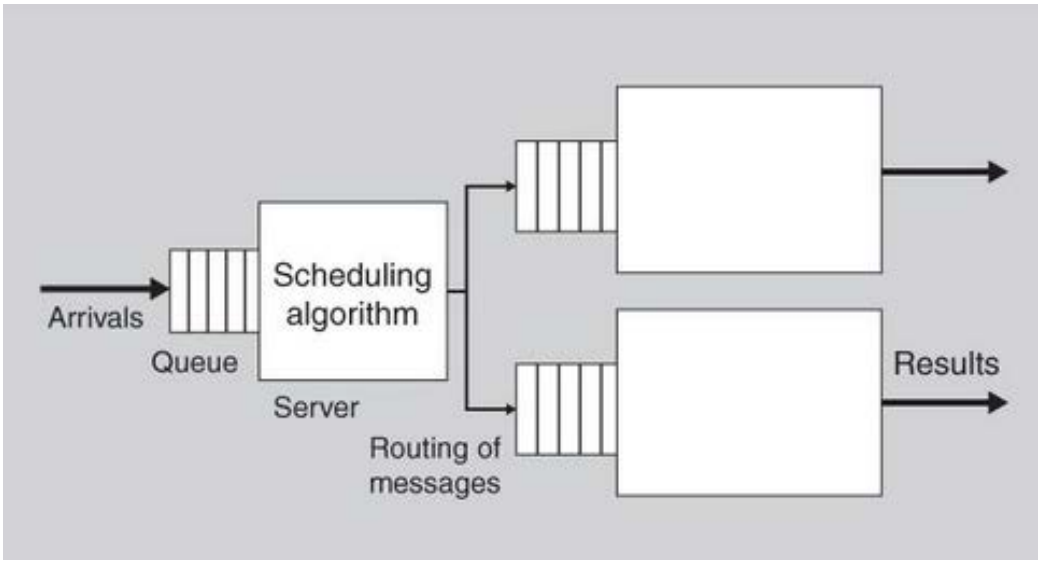
Атрибутите за качество имат модели за анализ.
Моделът за анализ е средство, с което се измерва количествено даден атрибут.
По-долу са разгледани примери на анализ за някои атрибути на качество.

Модел на опашките (queuing model)

Параметри:

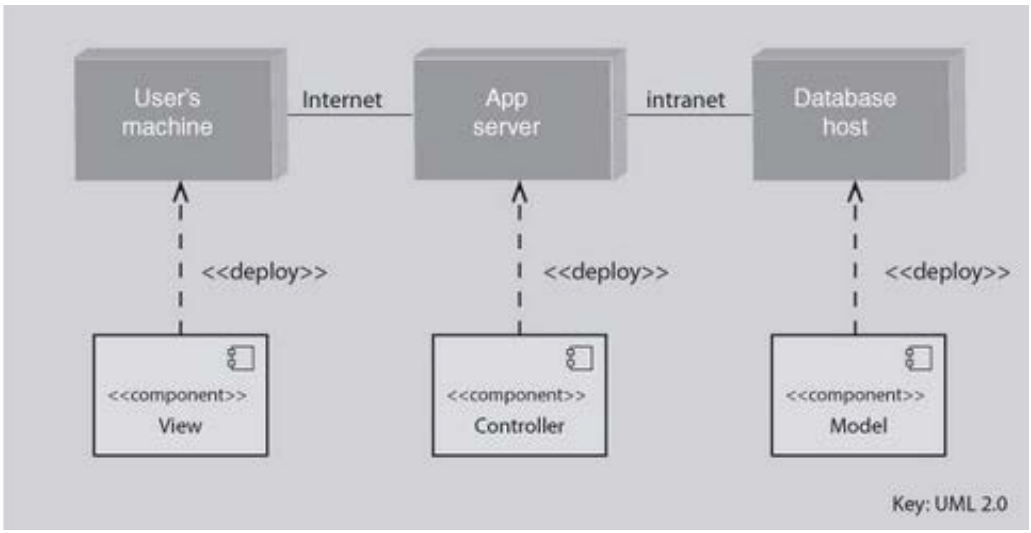
- скорост на пристигане на събития;
- модел на опашката;
- алгоритъм на режимиране;
- време за обработка на събитие;
- мрежова топология;
- скорост на мрежата;
- алгоритъм на маршрутизация.

Анализ на изпълнението

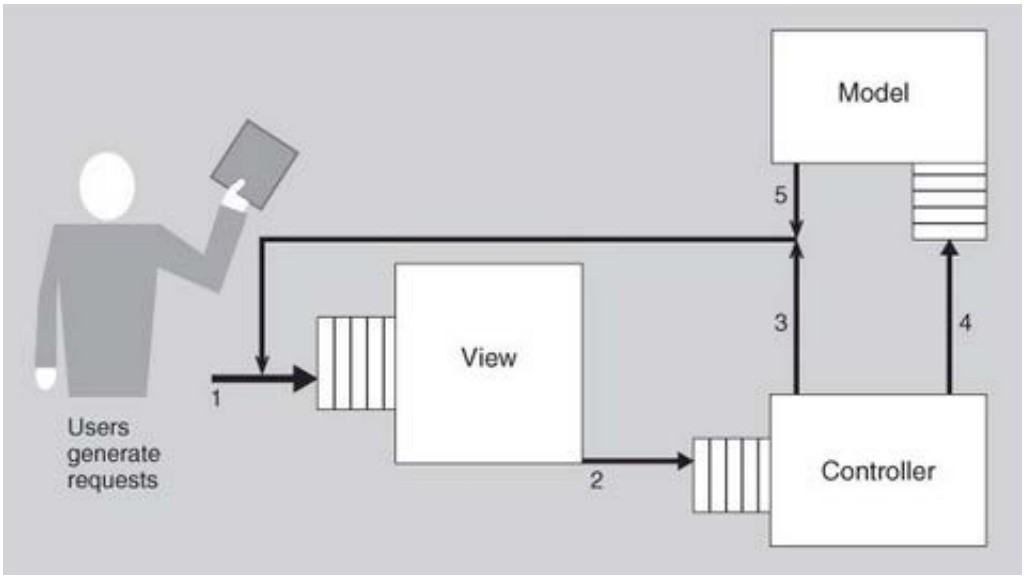


Фиг. 49 Модел на опашките – параметри и структура.

За да се анализира даден модел е необходима структура, на базата на която се извършва анализа.
Базовата структура в този пример е Модел-Гледка-Контролер, т.е. има връзка между процесите на модела, гледката и контролера. Моделът на опашките се използва за анализ на изпълнението.



Фиг. 50 Връзка между процесите на модела, гледката и контролера.



Фиг. 51 Модел на опашките за изпълнението в MVC.

Анализ на изпълнението – параметри:

- честота на пристигащите съобщения;
- време за обработка на съобщенията от гледката;
- брой на съобщенията, които гледката изпраща към контролера;
- време за изпращане на съобщенията от гледката към контролера;
- време за обработка на съобщенията от контролера;
- брой на съобщенията, които контролерът изпраща към гледката.
- брой на съобщенията, които контролерът изпраща към модела;
- размер на съобщенията, които се изпращат към модела от контролера;
- брой на съобщенията, които моделът изпраща към гледката;
- размер на съобщенията, които моделът изпраща към гледката;
- време за изпращане на съобщенията от модел към гледката.

Анализиране на наличността

Наличността показва за избраната архитектура колко процента от времето системата е достъпна.

наличност = MTBF / (MTBF + MTTR)

където

- MTBF = средно време между провалите (mean time between failure);
- MTTR = средно време за ремонт (mean time to repair).

В примера се използва брокер с тактики на излишества:

- активно излишество (active redundancy);
- пасивно излишество (passive redundancy).

Може да се използва и тактика за пулс с цел откриване на провали.

14.3. Атрибути за качество – проверочен списък

Проверочният списък помага на архитекта да си изясни пълнотата и точността на софтуерната архитектура.
Проверочният списък може да се прави за отделен атрибут на качество, за домейн, за специфичен аспект на отделен атрибут на качество.

14.4. Експерименти, симулации и прототипи

Експерименти, симулации и прототипи

Целта на експериментите, симулациите и прототипите е да се анализират алтернативите на архитектурата с по-коректни резултати и да се предвидят поведението и способностите на системата.

14.5. Анализ на различни етапи от жизнения цикъл на проекта

Различни анализи за различни етапи от цикъла.

Всеки анализ си има цена.

Различна вярност, увереност на анализа за всеки тип и етап.

Пример – Уеб-базирана система за конференции

Въпроси за анализ:

- Как ще повлияе замяната на централизирана база от данни с разпределена такава върху връзката с потребителите?
- Колко участника биха могли да достъпват едновременно от един сървър?
- Какво е забавянето между сървърите на базата от данни и конферентните сървъри?

Симулация: симулира се взаимодействието множество потребители със сървъра на конференциите.

Инструментите за мониторинг измерват натоварването на сървърите на базата от данни и тези на конференциите с различен брой потребители.

14.6. Анализ на различни етапи от жизнения цикъл на проекта

Различни анализи се провеждат на различните етапи от жизнения цикъл на проекта.

Всеки анализ си има цена и различно ниво на достоверност.

Етап	Форма на анализа	Цена	Достоверност
Изисквания	Аналог	Ниска	Ниска - висока
Архитектура	Чрез опит	Ниска	Ниска - средна
Архитектура	Проверочен списък	Висока	Средна
Архитектура	Аналитичен модел	Ниска - средна	Средна
Архитектура	Симулация	Средна	Средна
Архитектура	Прототип	Средна	Средна - висока
Реализация	Експеримент	Средна - висока	Средна - висока
Готова система	Инструменти	Средна - висока	Средна

Таблица 2. Видове анализи.

14.7. Обзор

Анализът позволява ранно предричане на стойностите на атрибутите за качество, което определя дали се удовлетворяват целите на атрибутите за качество. Някои атрибути за качество се анализират по-лесно от други.

15. Модул 15 Процеси

15.1. Архитектура в Agile процеси

Архитектура в Agile процеси

В днешно време е вече демоде безкрайният процес с късно производство
Agility е:

- Софтуерен процес задоволяващ бизнеса
- Отговарящ на изискванията на заинтересованите лица – сега, в момента

Манифест на Agile Software development

- Индивидуалност и взаимодействат пред процеси и инструменти
- Работещ софтуер пред дълги документации
- Връзка с клиента пред договорени уговорки
- Отворен за промени пред следване на план

Agile – принципи

Задоволява клиента – той работи за клиента, което е от първостепенна значимост за всеки бизнес

Парче по парче предоставя софтуер – софтуерът се разработва на етапи, като на всеки етап се показва резултат на клиента

Позволява промени на изисквания – ако клиента репи, че не иска това което му се представя на кой да е етап, той има възможност да промени изискванията си и съответно да се имплементира във следващ такъв

- Бизнес и софтуерни разработчици работят заедно
- Face-to-face разговори
- Работещ софтуер е най-важното!
- Простота, яснота

15.2. Agility и методи в архитектурата

Agility и методи в архитектурата

Къде е архитектурата в един agile процес ?

Как архитектурата се наглася според принципите на agile?

Кои методи допринасят за встъпването на архитектурата

Кое е крайната точка на яснота

Agility и методи в архитектурата

Документация на архитектурата и YANGI

- Базирана на изгледи и следствия (Views & Beyond)
- Ако информацията не е нужна, не отделяй внимание и ресурси да я документираш
- Принцип “пиши за читателя”
- YANGI – you ain’t gonna need it

Agility и методи в архитектурата. Еволюция на архитектурата

- Архитектурата е нужна и в agile процеси
- За клиентите е важно да виждат цялата картинка
- Пример: ATAM: Architecture tradeoff analyses method
- Дефиниране на качествени атрибути и анализа им с мерки

15.3. Кратък пример за Agile архитектура

Кратък пример за Agile архитектура

Разработка на система за конферентна зала – WebArrow

Те трябва да:

- предоставят real-time отражения
- работят за различен хардуер
- предоставят ниска латентност
- предоставят висока сигурност
- лесни за модифициране и интегриране

Кратък пример за Agile архитектура – конвенционални модели

Примерни модели: top-down vs. bottom-up

Top-down (голямата функционалност се разделя на по малки такива): дизайн, анализ на структурите за да “срещнат” измерителите на качествените атрибути

Bottom-up (разглеждат се първо значимите под функционалности): анализ на набор от имплементации

Кратък пример за Agile архитектура – решение

WebArrow създаде бързо груб анализ на софтуера

Имплементира PoC'с – Proof of Concepts, който да докаже идеята на дизайна с малък за това пример като имплементация. В случай, че тя работи успешно, се разработва в по голяма степен

Рефакторират код – сорс кода се преработва и се прави по качествен от към качествени атрибути като изисквания

Добавят код според изискванията

Еволюционен модел - процесът еволюира към по добра и по добра имплементация, както от към качество на имплементация на функционални изисквания, така и от не-функционални (качествени параметри) такива

15.4. Насоки за Agile архитектура

Ангажимент към успешните заинтересовани лица

Задоволяване нуждите на ЗЛ

Инкрементално и еволюционно израстване на системата

Итеративно системно разработване

Управление на риска (risk management)

Адаптивно към промяна

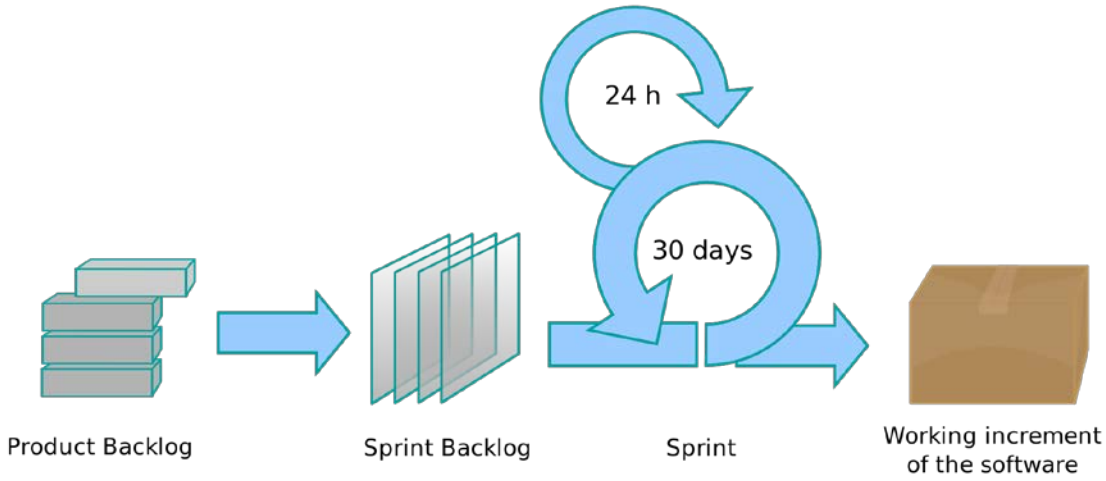
15.5. Водопад (Waterfall)

При метода на водопада имаме строго дефинирани граници на отделните фази при разработката на проекта. За всеки един не се позволява да се връща към предходния си, ако има такъв. Също така не се минава на следваща фаза, ако не се привърши всичко необходимо на предходна такава. Като се дефинират изискванията, тогава се

започва дизайна на архитектурата/системата, следван от имплементацията, тестването и съответно пускане в експлоатация и последваща поддръжка

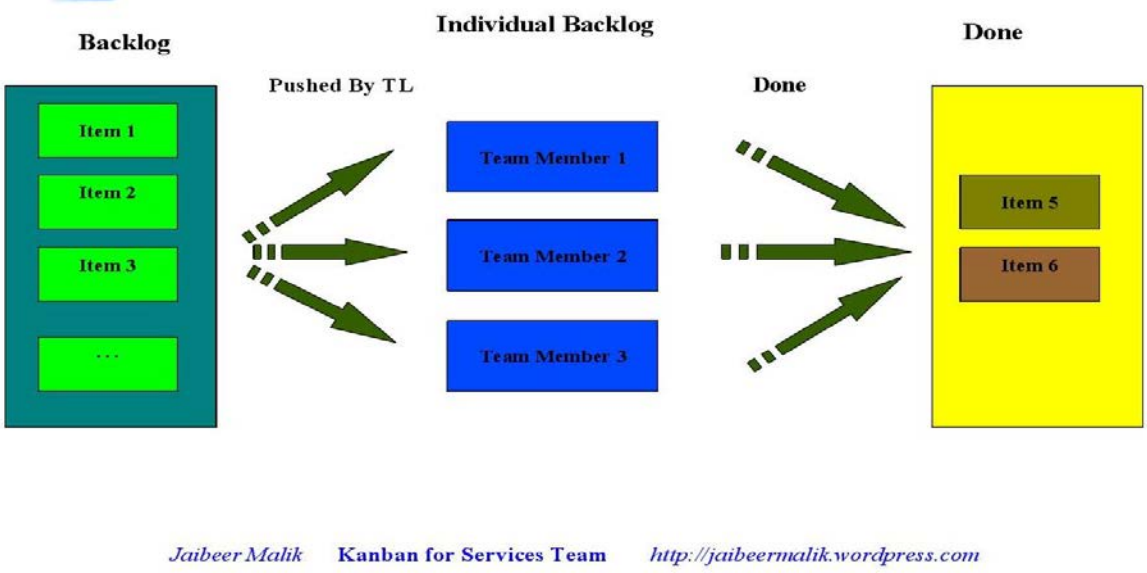
Този метод е подходящ за клиенти, които са наясно с функционалните си изисквания още в началния стадий на проекта. Няма гъвкавост и е статичен.

15.6. Scrum



При SCRUM методологията се набляга на дефинирането на кратки итерации, между 2 и 4 седмици, като на края на всяка итерация се показва готовият софтуер на клиента. Тази итерация се нарича Спринт (Sprint) в SCRUM. Също така, за да е ясно за всеки от екипа, кой по какво работи и с какви трудности се сблъсква, се провежда т.н. дневна среща („daily“) за около 15-20 минути. Задачите върху които се работи по време на спринта се приоритизират преди самото му начало и екипа работи по съответния приоритет, без промяна до края на спринта.

15.7. Kanban



При Kanban методологията, също като при SCRUM, се работи на кратки итерации. Разликата в случая е, че се задачите се приоритизират само от екипният ръководител и се предават на различните екипи за изпълнение.

15.8. Обзор

Agile движението е базирано на agile манифеста

Заложени са редица от стойностни и приложими във всички имплементации на agile принципи

Agile е подходящ за малки до средни проекти

16. Модул 16 Събиране на изисквания за софтуерна архитектура

16.1. Защо са важни ?!

Събиране на изисквания за софтуерна архитектура

“The two most important requirements for major success are: first, being in the right place at right time, and second, doing something about it”

- Рей Крос

Защо са важни ?!

Архитектурата е такава каквото е, за да удовлетворява изисквания

ASR – Architectural significant Requirements – изисквания касаещи задоволяване на архитектурата. Тези изисквания се събират от набор от заинтересовани лица, които са тясно свързани с функционалните и нефунционалните изисквания

Не можеш да направиш успешна архитектура без познаване на ASR

ASR's идва от :

- Чисто функционални изисквания
- Нефункционални такива
- Качествени атрибути

16.2. Събиране на ASR-и от документи на изисквания

Събиране на ASR's от документи за изисквания

Документи за изискванията са първоначалният източник за създаване на ASR's

Архитекти не трябва да чака крайният вариант на такива изисквания

Gathering requirements това е процес на събиране на изискванията на клиента. В последно време този термин се замества с „Извличане“ (eLicitation), което показва че в болшинството случаи, изискванията не са добре документирани и на архитектите се налага да разпитват клиента и да извличат недокументирани или неформализирани изисквания.

Такива източници са примерно:

- User stories
- Клиентски документи за описание на системата
- И т.н.

Ранни дизайн решения и изисквания, които оказват влияние

- Allocation of responsibility – планиране на
  - Степента на разпределяне на отговорности
  - Потребителски роли
  - Системни модули
  - Важни процеси – дефиниране на стъпки на въвеждане
- Coordination model
  - Свойства като: времеви рамки, консистентност, ниво на коректност, завършеност
  - Именуване на : външни елементи, протоколи, интернет конфигурации, middleware

- Data model:
  - създаване на процес на стъпки
  - Информационен поток
  - Домейн модел
  - Права за достъп

- Управление на ресурси:
  - Време
  - Конкурентост
  - Набор от активности
  - Графици

- Mapping among architectural elements:
  - Екипно планиране
  - Процесори – фамилия от процесори
  - Интернет конфигурации

- Binding Time Decision:
  - Задоволяване на гъвкави изисквания
  - Регионални ограничения

16.3. Събиране на ASR's чрез интервюта на ЗЛ

Събирането ASR's чрез интервюиране на ЗЛ

Резултат от такива интервюта включва:

- Усъвършенстване на системата
- Изчистване софтуерните изисквания
- Разбиране на архитектурните двигатели
- Упътване за разработването на прототипи и симулации
- Оказва влияние на реда на разработване на софтуера

Quality Attribute Workshop – стъпки

- Презентация и увод
- Бизнес / мисия
- Презентиране на архитектурният план

- Идентифициране на архитектурните двигатели

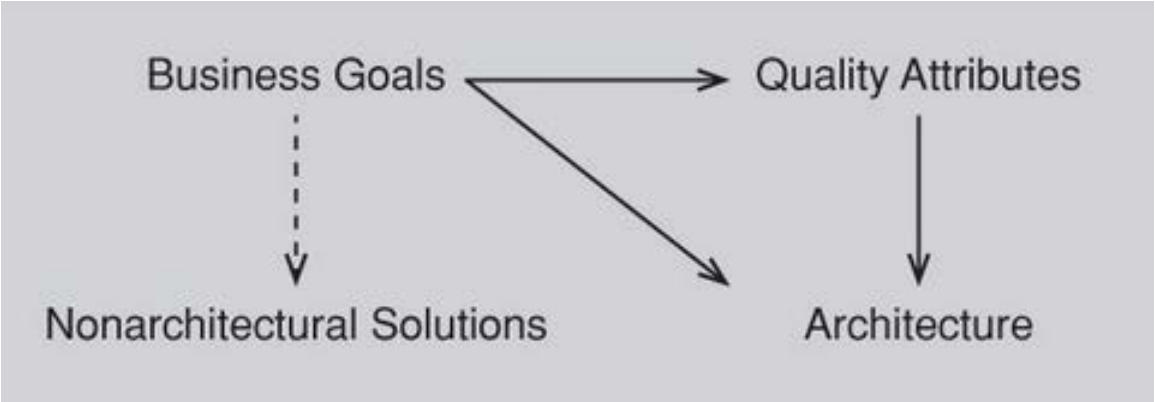
Quality Attribute Workshop – стъпки

- Сценарий за brainstorming
- Сценарий за затвърдяване на разработвания софтуер
- Сценарий за приоритизация
- Сценарий за усъвършенстване

16.4. Събиране на ASR's водени от бизнес целите

Събиране на ASR's водени от бизнес целите

Бизнес целите водят до:



фиг. 16.4.1

Бизнес целите водят до избирането на водещите качествени атрибути за системата, както и до архитектурен дизайн на системата. Този дизайн също се води и от избраните качествени атрибути

Събиране на ASR's водени от бизнес целите

Цели

- Покрива финансовите цели
- Покрива персонални цели
- Допринася за усъвършенстването на компанията
- Дава отговорности за служителите
- Дава отговорности на ЗЛ
- Менажира маркетинг позиция
- Подобрява бизнес процеси

Събиране на ASR's водени от бизнес целите

Сценарий – шаблон:

- Източник на целта

- Предмет на целта
- Обект на целта
- Обкръжение
- Цел
- Мярка на целта
- Произход и стойност

Събиране на ASR's водени от бизнес целите

Обзор на бизнес целите чрез PALM:

- Pedigreed eLicitation Method
- Pedigree - бизнес целта има ясен произход
- Използва вече описани стандартен списък
- Използва вече описаните сценарии

16.5. Структуриране на ASR's в utility tree

Структуриране на ASR's в utility tree

PALM- стъпки:

- Презентация за преглед, обзор
- Презентиране на Бизнес двигатели
- Презентация на архитектурните двигатели
- Извличане на бизнес целите
- Идентифициране на потенциални качествени атрибути от бизнес целите
- Заключително упражнение

Обзор

Архитектурата се тласка от изчистени бизнес изисквания
Изискванията към архитектурата се извлича от:

- Документация на изискванията
- Интервюта на ЗЛ
- Правене на Quality Attribute Workshop

17. Модул 17 Проектиране на архитектура

17.1. Жизнен цикъл

Жизнен цикъл

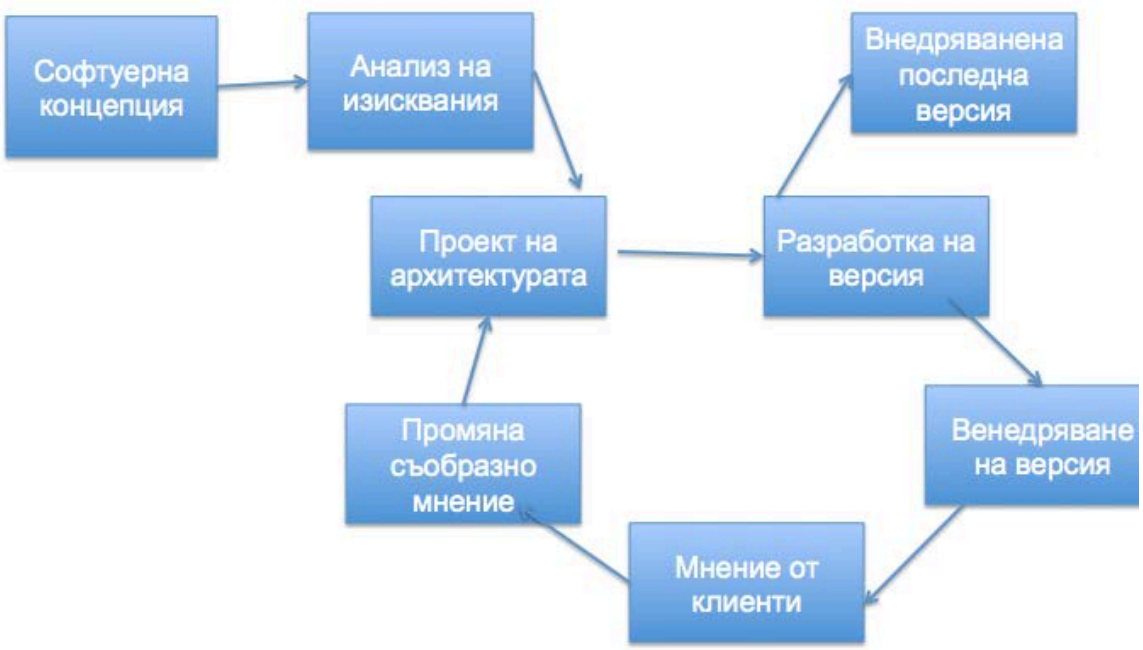
Къде е архитектурата в жизнеият цикъл на проекта ?

Печеливш модел: “Жизнен цикъл с еволюционно внедряване”

Затвърдяване чрез итерации

За всяка итерация се допитваме до клиента

Жизнен цикъл



Жизненият цикъл на проекта започва от анализа на изискванията. Те не е необходимо да са напълно изчистени и ясни при самото му инициализиране. Това може да се изчиства в процеса на работа. След първоначалният анализ се предприема пакетиране на първоначална версия на архитектурата, имплементира се, внедрява, тества и отново завъртваме цикъла. Докато не се имплементира последна приета от клиента версия

Жизнен цикъл – как започва ?

Започва при наличие на изисквания

Архитектурата се оформя от няколко (10-на):

- основополагащи функционалности
- Качествени атрибути
- Бизнес изисквания
- Др. Архитектурни драйвери

17.2. Стратегия за дизайн

Стратегия за дизайн

1. Идентифицират се целите на системата с най-висок приоритет
2. Тези цели се превръщат в
   - сценарии за употреба
   - Постигане на качествени свойство
   - От тях се избират най-влияещите за архитектурата
3. Това са архитектурните драйвери (drivers)
4. След това започва и проектирането
5. По време на което се:
   - Задават се въпроси
   - променят се изисквания
   - което довежда до и евентуална промяна на драйвери

17.3. Метод на проектиране, воден от атрибутите (Attribute-Driven Design ADD)

ADD е подход за проектиране. В него основна роля играят свойствата (атрибути) на качеството.

ADD е рекурсивен процес на дефиниране на архитектурата чрез прилагане на тактики за постигане на желани атрибути.

С ADD се получават първите нива на декомпозиция на модулите и другите структури.

Метод на проектиране, воден от атрибутите (Attribute-Driven Design ADD) – стъпки с пример

Примерна система за илюстрация на ADD:

- Продуктова линия за уреди за управление за гаражни врати (УУГР)
- Интеграция с Домашна информационна система (ДИС)
- ДИС:
  - Затваря/отваря гаражна врата
  - Диагностицира модула
- УУРГ :

- Затваря/отваря гаражна врата
- Задейства се с бутон или ДИС

ADD – входни данни

Набор от изисквания

- Функционални изисквания (use-cases)
- Функционални ограничения (constraints)
- Качествени свойства

ADD – изисквания на примера

Процесите, контролерите и устройствата в продуктовата линия се различават

Архитектура за специфичен продукт се получава от тази на продуктовата линия

В случай на пречка врата спира за най-много 100мс

Устройството се диагностира и достъпва от ДИС

ADD – стъпки

Избираем модул се декомпозира

Модулът се детайлизира:

- Избират се архитектурни драйвери
- Избира се архитектурен модел, удовлетворяващ драйверите
- Моделът зависи от тактиките с които ще се постигат АК
- Избират се типовете под-модули, необходими за постигането на тактиките

ADD – стъпки

Модулът се детайлизира

- Създават се под модули от идентифицираните типове
- Приписва се функционалност на под модулите в зависимост от use-case
- Създават се всички необходими структури
- Дефинират се интерфейсите на под-модулите
- Проверяват се и се детайлизират изискванията
- Ако всичко е налично се действа а по следващата декомпозиция

Минава се към друг модул, ако е необходимо

ADD – пример – стъпка 1

- В началото -
  - Цялата система е първия модул
  - Този модул разлагаме на под-системи
  - Под-системите се разделя на модули
  - Модулите се разлагат на под-модули
  - И т.н.
- В нашия случай – УУГВ
- Примерно изискване – УУГВ се интегрира с ДИС

ADD – пример – стъпка 2.1

- Избират се архитектурни драйвери
  - Изисквания с най-висок приоритет
  - При нас – 3-те дадени по рано
  - Пр. <100мс – изследва се механизъм на врата и скоростта на процесорите
  - Изборът на драйвери определя де-композиция на под-модули

ADD – пример – стъпка 2.2

- Избор на архитектурен модел
  - За постигането на всяко изискване има тактика
  - Конфигурация от тактики определя модел
  - Гледа се влияние между тактики - странични ефекти
  - При нас: *лесна промяна* и *бързодействие*

ADD – пример – стъпка 2.2

- Тактики за лесна промяна:
  - Локализиране на промените
  - Ограничаване на вълнообразни ефекти

- Отлагане на обвързването
- Поддръжка на семантична свързаност
- Скриване на информацията

ADD – пример – стъпка 2.2

- Тактики за бързодействие:
  - Намаляване изискванията на ресурсите
  - Управление на ресурсите
  - Управление на раздаването на ресурси
  - Увеличаване на ефективността

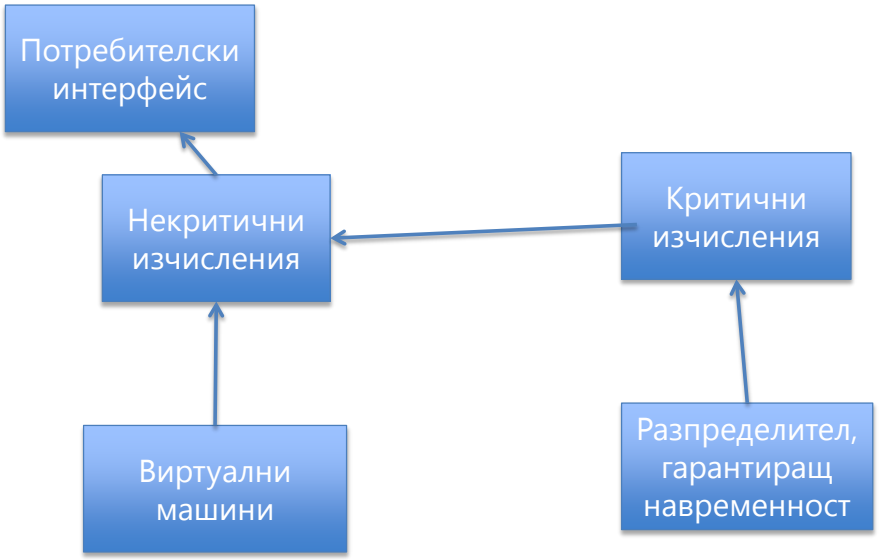
ADD – пример – стъпка 2.2

- Поддръжка на семантична свързаност и скриване на данните:
  - Отделят се в собствени модули функционалности:
    - Потребителски интерфейс
    - Комуникации
    - Сензори

- Увеличаване на ефективността:
  - Ефективни изчисления за критични активности

- Разумно разпределение на ресурси:
  - Навременно разпределени
  - Съгласно изискванията за критични изчисления

Архитектурен модел – типове под модули



Основните модули на системата са:

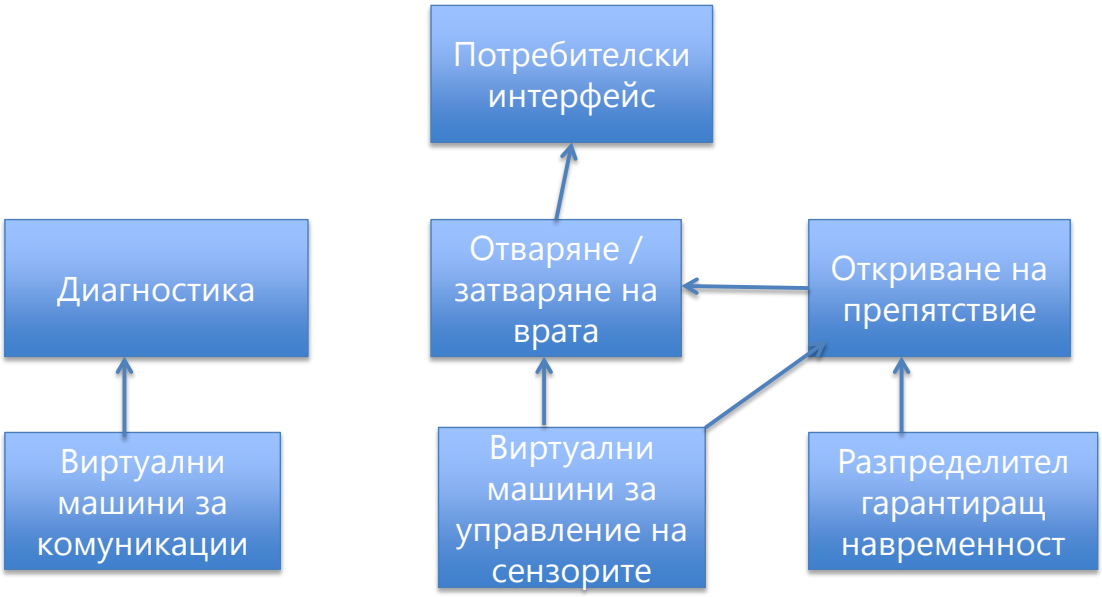
- Модул за не критични за системата изчисления – това са, като пример, статистики, от които може да се взаимства
- Модул за критичните изчисления – тук ще се съсредоточат процесите за затваряне на вратата, като пример. За този модул ще се отдели повече ресурси и внимание
- Модул “Разпределител, гарантиращ навременност” се грижи да предаде необходимите сигнали на модула “Критични изчисления” за гаранция на навременни операции
- Модул “Виртуални машини” – грижи се за виртуализация на сървърите, на които ще се внедрява софтуера

ADD – пример – стъпка 2.3

- Създаване идентифициран тип се създават под-модули
  - Критично изчисление:
    - Откриване на препятствия
    - Спиране на вратата
  - Не критично изчисление:
    - Отваряне/затваряне на вратата
    - Диагностиране

- Виртуални машини:
  - За комуникации
  - За работа със сензорите
- Потребителски интерфейс

ADD – пример – стъпка 2.3



ADD – пример – стъпка 2.3

- Приписване на функционалност
  - Всеки use-case -> модул
  - Резултат – сценарии за под модули
  - Възможно е в следствие да се премахват / създават под-модули
  - Изчиства се и данни на обмен между под модули

ADD – пример – стъпка 2.3

- Създаване на други структури
- Структури на декомпозиция – директно от метода
- Структура на конкурентно изпълнение
  - Паралелни активности
  - Синхронизация

- Идентифицират се deadlock, съперничества на ресурси и т.н.
- Дефинират се компоненти – модулите от декомпозицията
- Конектори – логическа връзка между модулите:
  - Синхронизация с
  - Стартиране
  - Спира
  - Комуникира

ADD – пример – стъпка 2.3

- Структура на внедряването
  - Необходимо поради наличието на различни хардуерни устройства
  - Декомпозират се логическите връзки
  - Техните части се разполагат върху различни процесори
  - Между тях се оформят информационни канали

ADD – пример – стъпка 2.4

- Дефиниция на интерфейсите на под-модули:
  - Услуги и свойства предлагани от модула
  - Ограничения от модула
  - Документират се

ADD – пример – стъпка 2.5

- **Проверка на декомпозицията:**
  - **Как функционалните изисквания се изпълняват от модулите**
  - **Дали изпълняват ограниченията?**
  - **Как се покриват сценариите за качество?**

ADD – пример – стъпка 3

- Рекурсивен ADD:
  - От стъпка 2 имаме под-модули за които имаме:
    - Списък с отговорности
    - Сценарий за употреба
    - Интерфейси
    - Сценарий за качество
    - Списък ограничения
  - Ако се налага е прави за под-модул рекурсивен анализ

Формиране на екипи

- След идентификация на няколко нива на декомпозиция
- Екип – модул
- Структура на екип – структура на декомпозиция
- Екип – със собствена експертиза

Създаване скелета на системата

- След
  - готова архитектура донякъде
  - сформирани екипи

  Се започва системата

Evolutionary deployment Life Cycle

- Така описаната скелетна система продължава и след внедряването на работна система
- Това се описва в Evolutionary deployment Life Cycle
- EDCL се интегрира успешно от големи компании

Обзор

ADD е метод на генериране и тестване начина мислене

Итеративен метод с еволюционен характер, където

- Избира е елемент за дизайн
- Разглеждат се изискванията и удовлетвореността им
- Създаване се и се тества дизайн

18. Модул 18 Документиране на софтуерната система

18.1. Архитектурно описание

Архитектурно описание

- Включва:
  - Диаграми
  - Нотации
  - Архитектурни views и view points
  - Трансформира архитектурна информация в стойностни, разбираеми модели
  - Описва функционални и нефункционални изисквания на системата
  - Организирани в Architectural Description Language (ADL)

Архитектурно описание - ADL

Описанието на архитектурата включва диаграми, използвани нотации, архитектурни гледки и гледни точки, преобразуване на архитектурата в стойностни, разбираеми модели, описание на функционалните и нефункционалните изисквания към системата.

Нотацията, която се използва е Architecture Description Language (ADL).

Архитектурата се представя за всички заинтересовани лица. Архитектът представя архитектурата на всички заинтересовани лица с помощта на ADL. Описанието на архитектурата спомага за формулирането на задачите за изграждане и валидация на системата. Тя е база за по-следваща реализация.

Описанието на архитектурата е в основата на създаването на прототипи на системата.

ADL – примери

- ACME – http://www.cs.cmu.edu/~acme/
- Repide - http://complexevents.com/stanford/rapide/
- Wright - http://www.cs.cmu.edu/~able/wright/

- Unicon - http://www.cs.cmu.edu/~Vit/unicon/reference-manual/Reference_Manual_2.html
- ByADL
- ABACUS
- И много други, домейн специфични

18.2. Употреба и аудитория на информацията на архитектурата

Употреба и аудитория на информацията на архитектурата

Документацията служи за множество цели. Тя трябва да е достатъчно ясна за всички заинтересовани лица. Архитектурата трябва да е разбираема от всеки нов член на екипа или заинтересовано лице. Архитектурата е база за анализ на бъдещата система. В нея са описани ограниченията и взетите решения.

Документацията на архитектурата се използва основно като средство за обучение, за комуникация между всички заинтересовани лица, за изграждане и анализ на системата.

Употреба и аудитория на информацията на архитектурата

Три основни случаи на употреба на документацията:

- Основен обучителен ресурс
- Основен елемент за комуникация между всички заинтересовани лица
- Основен елемент за конструкция и анализ на системата

18.3. Начини на документиране на архитектурата

Нотации на документиране на архитектурата

Различни в зависимост от тяхното ниво на формалност

Три основни категории на нотации:

- Неформални нотации
- Полу-формални нотации
- Формални нотации

Неформални нотации

Изгледите са изобразени използвайки общ, неспецифична диаграмна нотация

Семантика на описанието използва натурален език – не може да бъде анализиран

формално

Примерен инструмент - PowerPoint

Полу-формална нотация

Изгледите са изобразени със стандартна нотация

Описваща графични елементи и правила на конструкция

НЕ предоставя пълна семантична значимост на тези елементи

UML е чудесен пример: това е език за дизайн и описание на софтуерната система за представянето ѝ в различни типажа изгледи. Всеки от тях се концентрира към съответно заинтересовано лице

Формална нотация

Изгледите са изобразени с прецизна, целенасочена нотация

Семантика и синтаксис според контекста - езиците за описание силно зависят от контекста на проекта.

ADL – тип нотация за специализирани за конкретни архитектурни изгледи

18.4. Изгледи

Изгледи/Гледки

Архитектурата не може да се опише само и единствено в една посока или в един контекст. Тя е набор от системни елементи и отношенията между тях.

Гледките (представянето на архитектурата) се различават и разделят по типове. Всяка гледка има различна цел и аудитория.

Модулна гледка (Model view, изглед на модела)

Модулът е основна единица за реализация с набор от отговорности с висока свързаност. Модулът може да е клас, пакет, слой, аспект или всякаква декомпозиция на единица софтуер.

Елементите са модулите, представящи единица софтуер.

Връзките са от вида:

- част от (part of) – дефинира част от система;
- зависи от (depends on) – един модул зависи от друг;
- е (is a) – дефинира йерархия на наследяване.

Модулната гледка се използва за задаване на топологични ограничения, например за видимост на използването от други модули.

Целите на модулната гледка са да се изгради скелет на кода, да подпомогне анализа

на системата, да е база за планиране на разработката, да свърже реализацията с изискванията, да служи за връзка между структура и реализация, да свърже задачите с модулите.

Модулната гледка има следната структура: име, отговорности, видимост на интерфейсите, представяне на информацията, връзка с изходния код, тестова информация, информация за управлението, реализация на ограниченията, история на измененията.

- Име (Name)
- Задължения/ отговорности (Responsibilities)
- Видимост на интерфейсите (Visibility interfaces)
- Представяне на информацията (Information implementation)
- Връзка с изходния код (Source code unit mapping)

Гледки компонент-конектор (Component-and-Connector views), C&C

Гледката компонент-конектор (C&C – Component & Connector) представя елементите по време на изпълнение: процеси, клиенти, сървъри, база данни и т.н. Това са компонентите. Тя включва и начините на комуникация между компонентите.

Пример за такива гледки са клиент-сървър.

Компонентите имат интерфейси – портове.

C&C – елементи, връзки, ограничения, използване

Елементи:

- Компонентите (components) – имат портове, чрез които взаимодействат с другите компоненти.
- Конекторите (connectors) представят начините на взаимодействие между компоненти. Те имат правила за комуникация.

Връзки:

- Закачване (attachments) – портовете на компонентите са асоциирани с конекторите.
- Делегирането на интерфейси (interface delegation) е асоциацията на портовете с повече от един компонент.

Ограничения:

- Компонентите могат само да се свързват само с конектори – не и към други компоненти.
- Конекторът не може да съществува в изолация, без връзка с компоненти.

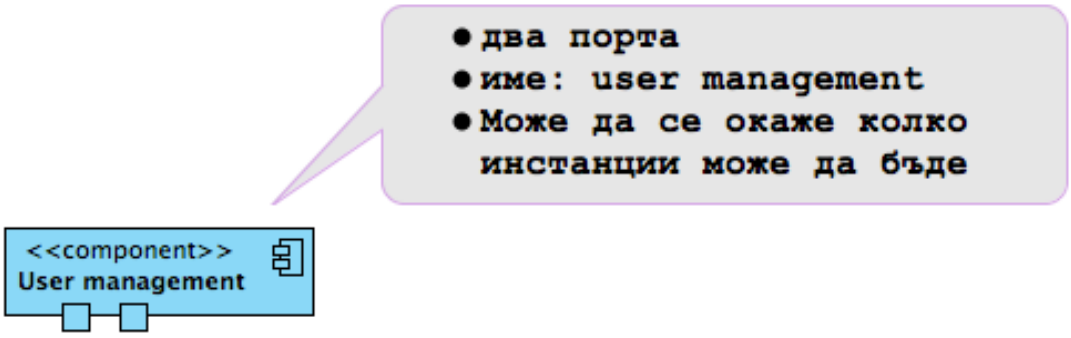
Използване:

- Показва как работи системата.
- Води разработката задавайки структурата и поведението в процеса на изпълнение.

C&C – свойства

- Надеждност (reliability)
- Изпълнение (performance)
- Изисквания към ресурсите (resource requirements)
- Функционалност (functionality)
- Сигурност (security)
- Конкурентост (concurrency)
- Изменяемост (modifiability)
- Ниво (tier)

C&C – примерна нотация



В този пример се илюстрира компонент с два порта за входни параметри, най-често различни по типаж и структура. Именуван е user management, което е подсказка за целта му – отговорностите

Гледки на разпределението (Allocation Views)

Гледката на разпределението описва връзката между софтуерните единици и инфраструктура на разработване и изпълнение. Това включва екипите на разработване, сървърите на изпълнение, операционната система и т.н.

Тази гледка съдържа елементите на инфраструктурата и елементите на софтуера.

Гледки на разпределението - елементи, връзки, ограничения, използване

Елементи:

Софтуерни елементи – софтуер със свойства, изисквани от инфраструктурата.

Инфраструктурни елементи – имат свойства, които са необходими на софтуера.

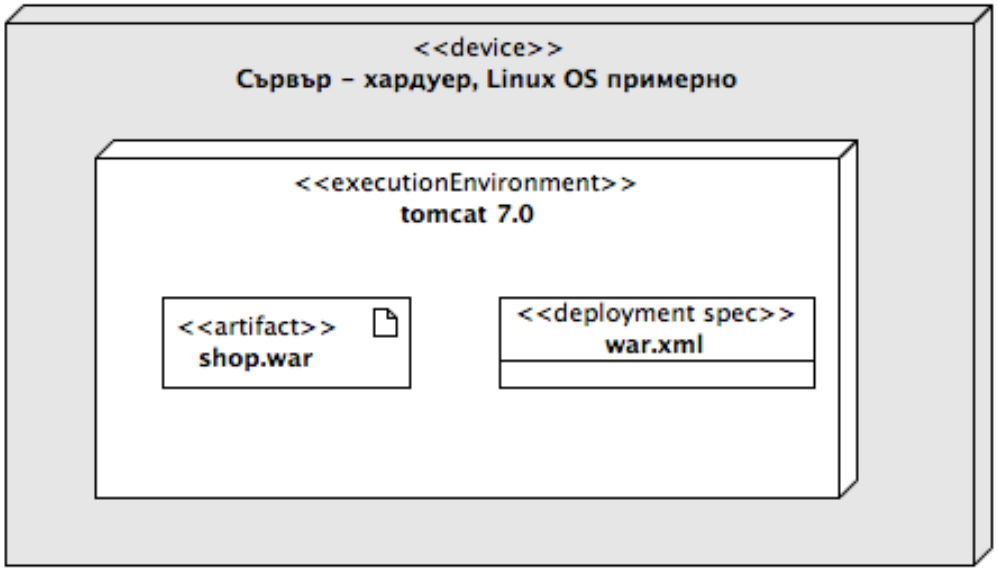
Връзки:

Разпределен към (allocated to) – софтуерната единица е свързана с инфраструктура или екип за разработка.

Ограниченията варират според конкретните гледки.

Използването на тази гледка е за целите на изпълнението, наличността, сигурността, безопасността, разпределението по екипи на разработка, конкурентния достъп и синхронизация.

Гледки на разпределението – примерна нотация и диаграма



На фигурата се илюстрира сървър, като хардуер, на който се инсталира inux като операционна система. На него се внедрява tomcat сървър като уеб апликационен такъв, в който се внедряват shop.war приложението с описание на внедряването в war.xml файла

Други гледки

Разгледаните до тук гледки са структурни гледки, но има и още много други видове като гледките към:

качеството (атрибутите за качество);

сигурността;

комуникациите;

изпълнението;

обработката на изключенията.

Гледките включват дефиниране на метрики и начините на достигането им.

18.5. Избор на изгледи/гледки

При избора на гледки трябва да се вземе в предвид хората и техните знания, приложимите стандарти, бюджета, времето, необходимата информация, заинтересованите лица, водещите атрибути за качество, размера на системата.

Минималният набор от гледки е:

- модулната;
- компонент-конектор;
- разпределение;
- по една за трите водещи атрибута за качество.

Избор на гледки – стъпки за избор на гледки

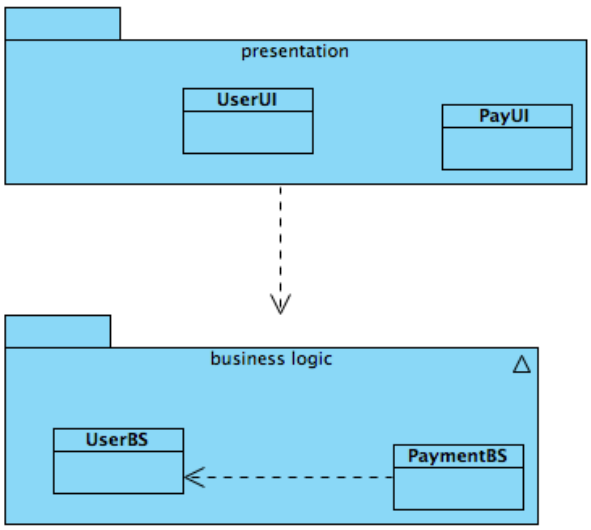
- Свържете се със заинтересованите лица и създайте гледка в табличен вид.
- Комбинирайте гледките.
- Приоритизирайте гледките.

18.6. Комбиниране на изгледи

Документацията на архитектурата е набор от взаимно свързани гледки.

Комбиниране на гледки става чрез асоцииране и задаване на елементите и връзките между свързаните гледки (например по контекст).

Комбиниране на гледки: декомпозиция по модули и използване (Module decomposition and Uses views)



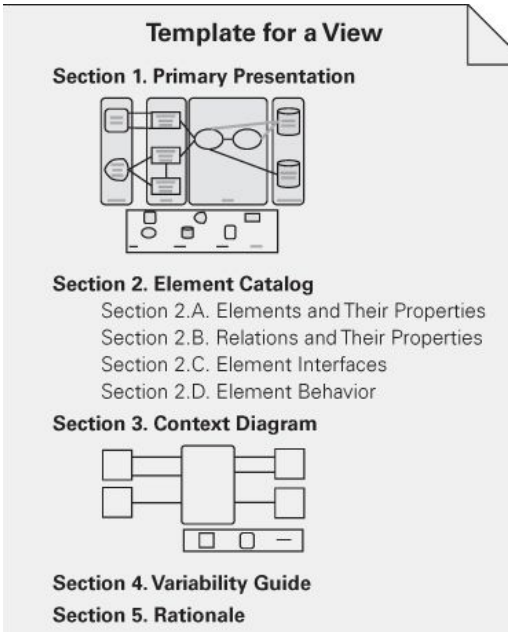
Комбиниране на гледки – примери

- Различни C&C гледки.

- Гледка на разпределение с ориентирана към услуги архитектура или с комуникациите. (Deployment view с SOA или communication processing view)
- Декомпозиция по модули с присъединяване на работата (Decomposition view с assign view).
- Декомпозиция по модули със слоеве(Decomposition view с layered view).

18.7. Изграждане на документацията

Изграждане на документацията
шаблон на гледка



Разбира се, не всички архитектурни описание могат да се водят от точно такава последователност и структура, но това е примерен шаблон, по който може да се документира софтуера.

В секция 1 се вписват съществото представяне на архитектурата и дизайна. Прикачват се различни изгледи, чрез които се разглеждат различни перспективи на архитектурата

В секция 2 се вписват детайлно елементите, които са използвани и проектирани в дизайна на софтуерната архитектура, техните интерфейси и поведение – отговорности, които носят със себе си

Основното представяне (Primary representation)

- Показва елементи и връзки на изгледа
- Съдържа информация по отношение на системата
- Съдържа важни, абстрактни, елементи
- Избягва подробности

- По често се представя графично, с диаграми

Каталог на елементите (The Element Catalog)

Описва най-малкото елементите изброени в основното представяне (Primary representation). Включва

- елементи и техните свойства
- Връзки и техните свойства
- Интерфейси на елементите
- Поведението на елементите

Диаграма на контекста (Context diagram)

Изобразява системата (или част от нея) и свързаните с тази гледка системи(и/или части от системата).

Контекстът се определя от това коя част от системата и как тя взаимодейства в тази гледка, с кои се свърза гледката – хора, други системи, физически обекти (сензори, контролни устройства и т.н.).

Ръководство за вариантността (Variability guide)

- Изобразява вариации на изгледа
- Всяка вариация е конкретен случай на използване
- Показва метрики или други полезни outputs
- Конкретизира се върху системата или част от нея

Рационалност (Rationale)

Дава яснота защо се използва тази гледка. Описва проблем; решение; потребителски случаи; метрики, ако е необходимо; последици.

18.8. Документиране на поведението

Изобразява как архитектурните елементи взаимодействат един с друг.

Причините за взаимодействие могат да са:

избягване на мъртва хватка;

завършване на задачата на системата;

заделяне на памет;

време за изпълнение;

конкурентно изпълнение на задачи;

връзки между различни задачи.

Документиране на поведението – нотации

Ориентирани към следата нотации. Следите описват взаимодействията между структурните елементи. Те описват резултата от системата в даден контекст, вадена

ситуация, в специфично състояние.

Примери за нотации:

- диаграми на потребителските случаи;
- UML диаграми на последователностите;
- UML комуникационни диаграми.

18.9. Документиране на архитектурата и качествените характеристики

Документиране качествените характеристики - начини

Описание на всяко проектно решение (образец) и асоциираните с него атрибути за качество. Например:

- клиент-сървър: добра за мащабируемост;
- ориентирана към услуги архитектура: добра за модулност.

Индивидуални структурни елементи, реализиращи услугите.

Атрибутите за качество въвличат със себе си и свой „език" . Например:

- сигурност – нива на сигурност, удостоверяване на потребители, защитни стени и т.н.
- изпълнение – мъртва хватка, буфер и т.н.

Архитектурните документи задават изискванията и начините на постигането им. Например, „наличност" се в документа и къде и как се използва.

Всеки атрибут за качество има своите заинтересовани лица.

18.10. Документиране на бързо развиващи се архитектури

Документацията се развива постепенно и еволюира след всяка итерация, като итерациите са от вида:

- скелет, доказателство на концепцията;
- реализация;
- документиране.

Всеки нов важен модул се документира в процеса на разработка.

Обзор

Документацията е насочена към определена аудитория.

Архитектът знае коя част от системата е добре да се документира.

Различните видове гледки и представянето им, спомага за по-доброто разбиране на системата.

19. Модул 19 Архитектура, внедряване и тестване

19.1. Увод

Архитектурата е само началото към една цел – софтуерен продукт/услуга
Но как тя се свързва със следващите фази на имплементация в този проект ?

Архитектурата има връзка с

a. имплементацията (и обратно)
b. тестването (и обратно)

Както бе обяснено за жизнения цикъл на проекта, архитектурата е самата инициализация на проекта, непосредствено след изчистване на функционалните и нефункционалните изисквания. Тя има тясна връзка с последващата я фаза – реализация/имплементация, която се прилага на дизайна, правилата, описани от архитектурата, а от там и самото тестване на приложението – създаването на различни видове тестови процедури

19.2. Архитектура и реализация

Архитектурата служи за макет на реализацията.
Архитектурата и реализацията трябва да са съгласувани.
С развитието на проекта се наблюдава и развитието на първоначалния замисъл на елементите, на декомпозицията, на компонентите и конекторите, на връзките и взаимодействията и т.н.

Тактики за съгласуване на архитектура и реализация
Вграждане на проекта в кода.
Ключовата задача на реализацията е правилно да изпълняват „рецептите" на архитектурата.
Периодичните проверки имат за цел да осигурят, че реализацията е по зададените архитектурни концепции.

Тактики за съгласуване на архитектура и реализация
Рамки (Frameworks)
В софтуерна система се използват софтуерни единици (модул, сорс код, библиотеки) с многократна употреба.
Класовете са подходящи за специфичният домейн на софтуерната система.
Софтуерните единици могат да са малки и/или големи.
Рамката може да не предоставя всички необходими задачи, а да се допълва.
Предимства:

- пести се време;
- може да решават сложни задачи;
- намаляват се възможностите за грешки.

Тактики за консистентност на архитектура и имплементация
Шаблони от код.
Шаблонът е структура, реализираща специфична архитектурна функционалност.
Генераторите на код предоставят генериран скелета от код на системата.
С шаблоните софтуерният инженер донаписва кода.
Шаблонът може да се задава и чрез диаграми, например с UML диаграми на класовете.

Следене на съгласуването на кода с архитектурата.
Кодът без надзор може драстично да се отдалечи от архитектурната идея.
Доброто управление и дисциплина в процеса на разработка предотвратяват „ерозията“ на архитектурата.
Добри практики за съгласуване на реализацията с архитектурата са:

- синхронизацията при всеки жалон;
- синхронизацията при кризисни ситуации;
- синхронизацията при нова версия.

Архитектура и тестване
Тестът е процес на потвърждаване на правилността на реализацията на изискванията към софтуера.
Архитектурата играе важна роля в тестването.
Тестването е по-евтино, ако е предвидено още на архитектурно ниво.

Тест – ниво и архитектурна роля

- **Тестване на модули (Unit testing)**

- Тестване на модули (Unit testing)

  Тестване на модулите.

  - Тестовете се пускат за специфична част от кода.
  - Зависимите модули, с които тестваните модули взаимодействат, се представят с „тапи“ (stub)
  - Тестовете, най-често, се пишат от самите разработчици.
  - Тестовете се пишат преди реализацията (Test – driven development).
  - Тестовете са на ниво клас, слой, част от слой.

- Тестване на модули (Unit testing) и Архитектура

  - Архитектурата дефинира модула и отговорностите му и изискванията към него.
  - Изискванията за изменяемост също могат да се тестват с тестването на модула.

  Тест – ниво и архитектурна роля

- Интеграционен тест (Integration test)

Тест – ниво и архитектурна роля

- Интеграционен тест (Integration test)
  - Тества как различни unit работят заедно
  - Нямаме „тапи“ (“stub”)
  - Тестване както unit-та, така и неговите зависимости
  - Използват се интерфейси в тестването
  - В края на интеграционното тестване се доказва:
    - Парчетата код – units – работят заедно
    - Верност на system-wide функционалност

Интеграционният тест проверява как различни модули работят заедно без тапи. Тества се както модула, така и неговите зависимости, като фокусът е върху интерфейсите.
С интеграционното тестване се доказва, че парчетата код (модулите) работят заедно и постигат функционалността на системата.

Видове интеграционни тестове:

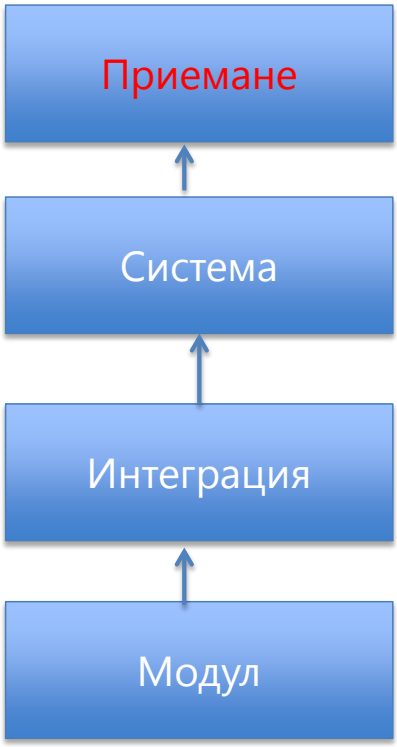
- Тестването на системата включва всички елементи на системата - софтуер, хардуер, инфраструктура.
- Интеграционно тестване със системи на трети страни.

Интеграционен тест и архитектура.
Може да се планира кога и как се интегрира системата.

Гледката на използване спомага да се разграничат интеграционните тестове.
Използват се дефинираните интерфейси в архитектурата.
Тестват атрибути за качество по време на изпълнение като изпълнение, надеждност.

Тест – ниво и архитектурна роля

- Тест за приемане (Acceptance test)



Тест – ниво и архитектурна роля
Тестът за приемане се извършва от потребители на система, като има два вида: алфа и бета тестове. Алфа тестовете се извършват от разработчиците. Потребителите нямат ограничения. При бета тестовете има ограничения и уговорки с потребителите, но тестовете се извършват при потребителите.

Приемен тест и архитектура (Acceptance test)

Специален вид тестове са стрес тестове с голямо натоварване, с акцент върху управлението на ресурсите, с атаки на сигурността и т.н. Тестват се мащабируемостта, изпълнението, наличността.

Тестове черна кутия и бяла кутия (Black-Box & White-Box тестове).

- При тестове черна кутия:
- Не се знае вътрешната реализация;

- Известни са само изискванията към софтуера;
- За архитектурата това е документираната библиотека от изисквания.

При тестовете бяла кутия:

- Използва вътрешната структура и елементи за тестване;
- Контролира потока на данните, алгоритми и т.н.
- За архитектурата може да се тестват атрибутите за качество – изпълнение, наличност, надеждност и т.н.

19.3. Архитектура и тестване

Ролята на архитекта в тестването

- Подпомага тестването;
- Проектира архитектура, подлежаща на тестване;
- Работи с екипа за тестване;
- Осигурява достъп до изходния код;
- Дава възможност за контрол на системите при тестване;
- Осигурява поддържка на множество версии на софтуера.

19.4. Обзор

Архитектурата играе важна роля, както в тестването, така и в реализацията.
По време на тестване, архитектурата дефинира какво се тества и на коя фаза.
По време на разработка, се следи дали софтуерът се съгласува с архитектурата.

20. Модул 20 Реконструкция (възстановяване) и съответствие на архитектурата

20.1. Процес на реконструкция на архитектурата

Дотук, архитектурата беше разглеждана от самото начало на проекта, но какво да се прави с код, който е получен наготово? Как този код да се управлява еволюционно? Как да се управляват атрибутите на качеството? Как да се документира кода?

Сега ще тръгнем по обратния път като започнем с готовия код.

Как се управлява еволюционно ?

Как се управляват качествените атрибути ?

Как да се документира ?

Процесът на реконструкция на архитектурата извлича модулите, функциите, файловете, обектите от кода на системата. Това е итеративен процес на реинженеринг. В него се използват голям набор от инструменти.

Фазите на този процес са:

- Извличане на гледката (Raw View Extraction)
- Конструкция на модела от данни;
- Синтез на гледката;
- Анализ на архитектурата.

Прилага се Реверсивен инженеринг (Reverse engineering) като практика. Това е е метод, който се прилага на вече разработен и имплементиран софтуер. Със него можем да дефиниране основните софтуерни елементи, които се използват в системата. Тези елементи се разглеждат първоначално на най-високо ниво на абстракция, от което постепенно се минава на по конкретно такова и съответно разделянето на други такива елементи. Така рекурсивно се стига до ниво, което е полезно за разбирането как работи една непозната система

20.1.1. Извличане на груба информация (Raw View Extraction)

Извлича се базова, “груба” информация за архитектурата.

Извличането на тази гледка гледката очертава най-общо архитектурата. Източниците на информация са изходния код, скриптовете за изграждане, следите на изпълнението – анализ на артефактите.

Целта е да се извлекат различни гледки, като се даде отговор на различни архитектурни въпроси.

От изходните артефакти се получават елементите (файлове, функции) и връзки между тях. Последните отразяват кой, кого и как използва.

Връзки между елементи се представят във вид на граф. Например, по извикванията се строи граф на извикванията, по включванията се изграждат зависимостите между

модулите, по достъпите за четене и писане се очертава използването на данните и т.н.

Видове инструменти за реконструкция на гледки

Разпознавателите (parsers) генерират гледка към вътрешното представяне на кода.

Анализаторите на абстрактното дърво на синтаксиса (abstract syntax tree analyzers) са като разпознавателите, но генерират явно дървовидно представяне.

Лексическите анализатори (lexical analyzers) анализират лексически значимите елементи като поредици, маркери, образци.

Профилаторите (profilers) събират информация за кода по време на изпълнение.

Инструментариума за код (code instrumentation tools) се използва по време на тестване. Позволява да се добавя код.

Конструкция на модела на данни

В тази фаза, извлечената информация от кода се оформя в стандартен формат. Информацията се използва за реконструкция на модела на данни на абстрактно ниво. Например, премахват се частните извиквания на методите и остават само публичните такива.

20.1.2. Процес на реконструкция на архитектурата - фази

Интерпретатор / Parses /:

- Генерира вътрешна репрезентация на кода
- Генерира изглед /view/

Анализатор на абстрактен синтаксис (Tree Abstract Syntax Tree Analyzers):

- Подобно на Интерпретатор / parses /
- Генерира експлицитно дървовидно представяне

Лексикографски анализатори (Lexical Analyzers)

- Анализира лексикално значими елементи – стрингове, токъни, шаблони (patterns & matches)

Профили (Profiles):

- Дава информация за кода по време на работа

Инструменти за работа с кода (Code instrumentation tools):

- Приложимо в тестването
- Може да добавя код към вече генериран такъв

Конвертиране на грубо извлечената (exctracted) информация в стандартен вид

Тази информация се използва за реконструкция на базата данни

Наблягане на ниво на абстрактност, например - Премахване на частните (private) извиквания на методи и оставяне само на публичните

20.1.3. Синтез на изгледа

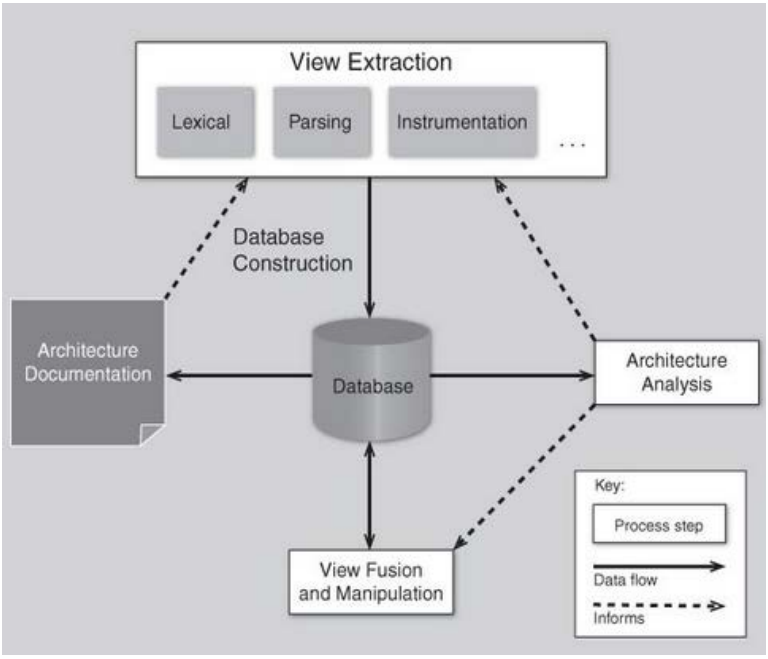
Комбинира различни изгледи на информация

Синтез на гледката

В тази фаза се комбинират различните гледки към информацията. Например, статичната гледка се извлича от кода, динамичната от изпълнението, и двете гледки се комбинират в гледки за декомпозицията по модули и използването им.

Целта е да изработват хипотези за да се визуализира архитектурата. Последната се показва от различни гледни точки.

20.1.4. Архитектурен анализ



20.2. Анализ на архитектурата и установяване на отклоненията

Анализ на архитектурата и установяване на нарушителите

Тук идеята е, че архитектурата и реализацията вървят ръка за ръка. Трябва да се избягва несъгласуването им.

Трябва да се провери правилността на кода от архитектурна гледна точка.

Тактики за запазване на съответствие между сорс код и архитектура са

- Съответствие чрез конструкция

1. Изгражда се скелет от модули или се генерира този скелет от диаграми.

2. Генерират се UML диаграми (UML Unified Modeling Language – унифициран език за моделиране)

- Съответстване чрез анализ

1. Осигурява съответствие чрез анализ
2. Реверсивен инженеринг (Reverse engineering)

20.2.1. Реверсивен инженеринг (Reverse engineering)

Това е процес на анализ на системата или на част от нея. Резултатът е по-абстрактно представяне на системата. За тази цел се използват различни гледки: декомпозиция по модули, компоненти и конектори, различните архитектурни елементи и т.н. Резултатът е по голяма абстракция на системата чрез различни гледки/изгледи:

- Декомпозиране по модули (Module decomposition)
- Компоненти и конектори (Components and connectors)
- Различни архитектурни елементи и т.н.

Мотивацията за реинженеринга е представянето на интерфейсите/контрактите, подобряването на документацията, пренаписването на остарялата система, модернизацията на софтуера, обучение.

20.2.2. Насоки и добри практики

Винаги трябва да има определена цел за възстановяване на архитектурата.

Винаги да се започва от най-високото ниво на абстракция.

Нужната информация трябва да се идинтифицира.

20.3. Обзор

Реконструкцията на системата може да докаже, че системата е реализирана по предназначение.

Реконструкцията на системата позволява да се модернизират на остарелите системи и да се разбират вече съществуващи такива.

21. Модул 21 Оценка на архитектурата

21.1. Фактори за оценка

Оценка на архитектурата

Анализът лежи в основата на качеството на архитектурата

Оценяващите методи подсигуряват, че архитектурата

- Е направена по предназначение
- Покрива първоначалните изисквания

Фактори за оценка

Оценка от Архитектурен Дизайнер

Дизайн решения се тестват чрез анализ техниките на качествените атрибути

Зависимости за определяне степен на анализ:

- Важността на дизайн решението
- Бройка на потенциални заместители
- От последствия

Оценка от колеги/Peer review/

Извършва се така както се провежда преглед на код. Може да се провежда по всяко време/фаза на проектиране на архитектурата.

Стъпките на прегледа са:

1. Определят се атрибутите за качество, по които ще се проведе прегледа.

2. Архитектът определя частта от архитектура, която ще се преглежда.
3. За всеки сценарий на атрибут за качество архитектът/проектантът обяснява как се постига.
4. Потенциалните проблеми се разкриват.

Оценка от Външни специалисти

Външните специалисти могат да дадат обективно мнение на архитектурата.

“Външни специалисти” могат да са относно бизнеса, фирмата или екипа, стига да познават приложната област.

Може да се разглежда цялата архитектура или само част от нея.

Критерии/въпроси при оценяване

- Какви са артефактите, подлежащи на оценка?
- Каква е целта на оценяването?
- Кои заинтересовани лица участват?
- Какви са бизнес целите?

21.2. Метод на анализ на архитектурните компромиси (Architecture TradeOff Analyses)

Методът на анализ на архитектурните компромиси (Architecture TradeOff Analyses – ATAM) оценява архитектурата по целите и постижимостта на параметрите на атрибути за качество.

Методът се направлява по бизнес цели и атрибути за качество. Той извежда потенциалните рискове, като дисбаланс между бизнес целите и архитектурата, и погрешно взетите решения

Защо се прилага ATAM и какви предизвикателства има?

Въобще, може ли системата да бъде анализирана?

Колко често анализът трябва да се извършва?

Изискванията към системата са за изменяемост, изпълнение, сигурност, взаимодействие, преносимост и т.н.

За всеки атрибут за качество трябва да се определи какъв процент от него е достигнат.

От къде екип за анализ взима метриките за оценка?

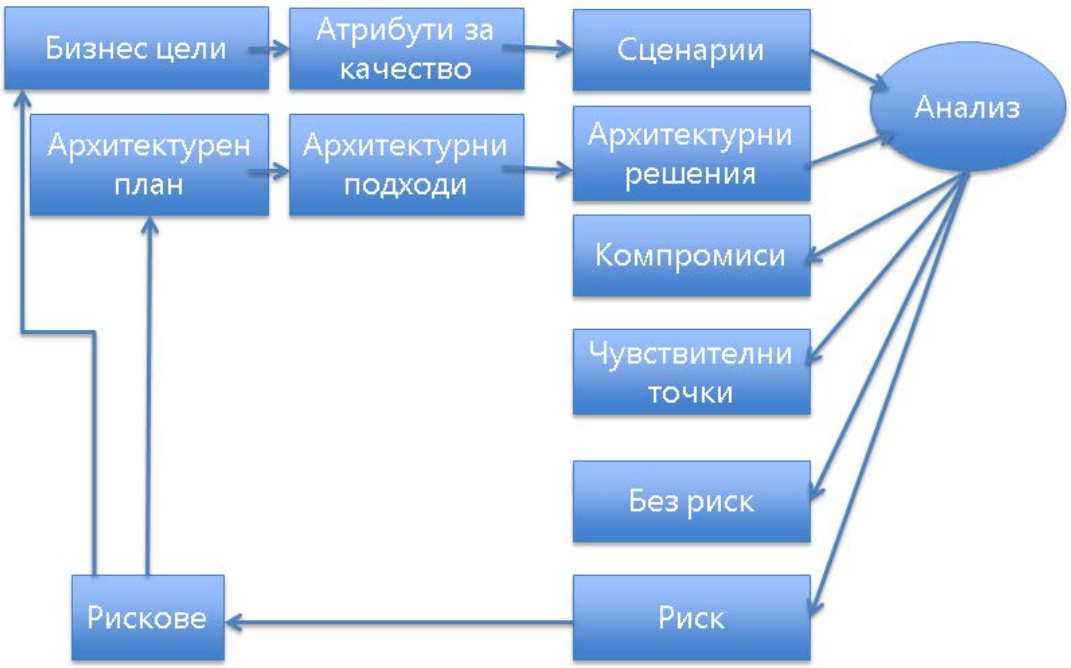
ATAM – участници

Ядрото на екипа за оценка е външен за проекта екип от 3 до 5 специалиста с роли: лидер, сценарист, изпълнител, задаващ въпроси.

Участват още всички, които взимат решения по проекта. Те комуникират с разработващият екип и имат власт над проекта.

В екипа се включват и заинтересованите лица по архитектурата (architectural stakeholders).

ATAM - поток на изпълнение



ATAM - стъпки

1. Представяне на ATAM
   - Описва се метода
   - Представят се участниците
   - Какво се очаква като резултат
2. Представят се бизнес драйверите
   - Описват се бизнес целите
   - Описват се бизнес двигателите
3. Представя се архитектурата
   - Описва се архитектурата
   - Как тя задоволява архитектурните цели
4. Идентифициране на архитектурните подходи
   - Идентифицират се подходите
   - Не се анализират
5. Генериране на дърво на качествени атрибути
   - Типизират се качествените атрибути

- Конкретните КА се композират в дърво
- Приоритизират се

6. Анализ на архитектурните подходи
   - Анализират се КА с най-висок рейтинг
   - Идентифицират се рискове, tradeoff points
7. Brainstorm и приоритизиране на сценариите
   - Между анализ екипа
   - Избират се ефективен набор от сценарии за работа
8. Анализ на архитектурните подходи
   - Анализ се сценариите с висок рейтинг
   - Те са основата на архитектурата и резултата
   - Отново се разглеждат рискове, чувствителни точки
9. Презентиране се резултатите
   - На базата събрана от ATAM анализа

ATAM – отчет

Съдържанието на отчета е:

- Набор от архитектурни подходи.
- Дърво на използване (utility tree) – дърво от атрибутите за качество.
- Набор от сценарии за постигане на архитектурата.
- Набор от сценариите за постигане на атрибутите за качество.
- Идентифицирани рискове.
- Идентифициране на не рискове.

ATAM – ползи

Методът е фокусиран върху приоритизираните атрибутите за качество. Атрибутите за качество се постигат на стъпки. Подобрява се документацията на архитектурата. Идентифицират се рисковете сравнително рано.

21.3. Лека форма на оценка

В ATAM обикновено са включени 20-30 човека, а това струва пари, време и ресурси. Не винаги е оправдано провеждането на такъв тежък анализ, но софтуерът се нуждае от анализ и оценяване.

Ето защо често се прилага олекотена форма на оценка (Lightweight Architectural Evaluation Method) ра по-малки и не толкова рискови проекти.

Извършва се по-бързо като стъпка 1, представянето на ATAM, се изпуска; стъпки от 2 до

7 най-много за 5 часа общо се провеждат; стъпки 7 и 8 се изпускат; стъпка 9, представяне на резултатите, се извършва за около половин час.

21.4. Обзор

Ако архитектурата е важна за бизнеса и софтуера, то тя трябва да се оценява. За това се използват доказани подходи и методи. ATAM е един от тези методи. Има и олекотена версия на ATAM.

22. Модул 22 Мениджмънт и управление

Разглеждат се темите по отношение на управлението на проекта, начини на управление и вземане на решения, зависещи най-вече от целите на бизнеса. Резюмират се водещите фактори – пари, време, ресурси – които основно влияят на имплементацията на проекта

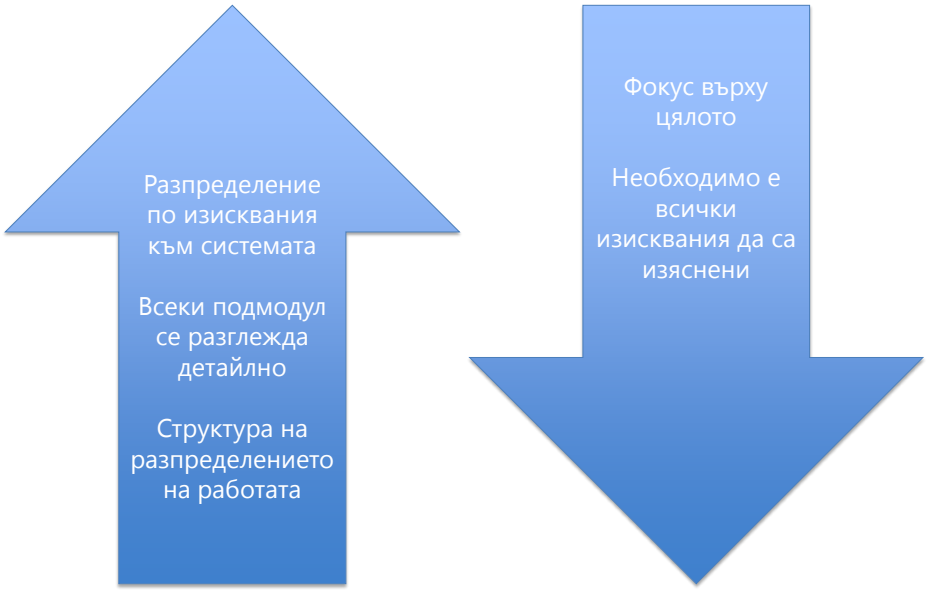
22.1. Планиране

Планирането се извършва през целия жизнен цикъл на проекта. То определя разходите на време, пари, нужните ресурси, дългосрочни и краткосрочни цели.

Дава идея за:

- Време – както в краткосрочни, така и в дългосрочни планове
- Пари – за отделни ресурси (работна ръка, хардуер, софтуер, лицензи, курсове и т.н.)
- Ресурси
- Цели - Краткосрочни срокове
- Цели – Дългосрочни срокове

Планиране Отгоре-Надолу спрямо Отдолу-Нагоре (Top-Down vs. Bottom-Up)



При Отгоре-Надолу (top-down)всяка функционалност се подразделя на под функционалности до момент, в който може да се дефинира единица време за внедряване (implementation). Добър инструмент в случая е използването на WBS – Work Breakdown Structure, чиято цел е подразделянена модули на ситемата, рекурсивно, до момент на яснота

Отгоре-Надолу и Отдолу-Нагоре (Top-Down Bottom-Up) – заедно

В този случай може да се подберат първо всички важни за системата функционалности, да се подразделят чрез top down метода и да се приложи съответната внедряване. Този хибрид допринася за бизнеса при някои случаи, в които бързи решения са от значение

22.2. Организиране

Организиране

Организиране на основни елементи в проекта, свързани с работата и управлението

Организация на

- Екип – брой хора, знания
- Разделяне на отговорности между мениджмънт и архитект
- Планиране на глобално ниво

Организация на екип

1. Разпределяне на отговорностите за модулите в проекта.

2. Разпределяне на функциите

- Кой пише документацията?
- Кой е отговорен за тестването?
- Кой е отговорен за управлението на конфигурациите?
- Кой ръководи срещите, плановете и други?

3. Типични роли в екипа:

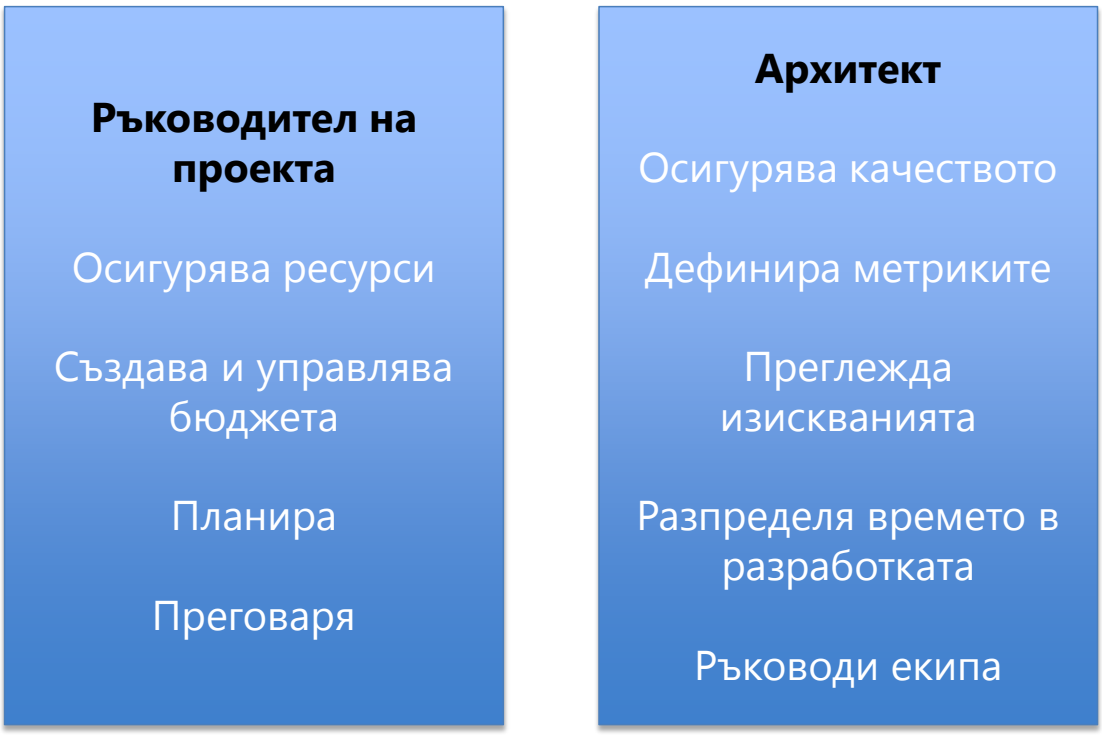
- ръководител на проекта
- софтуерен архитект
- разработчик
- отговорник за конфигурациите
- ръководител на тестването на системата

Разпределяне на отговорности между мениджър на проекта и архитекта

Основополагащата е връзката между ръководителя на проекта и архитекта. Те трябва да работят заедно за една и съща цел.

Ръководителят на проекта координира външните дейности на проекта, а архитектът – вътрешните.

Разпределяне на отговорности между мениджър на проекта и архитекта



Планиране на най-високо ниво

В този процес се разглеждат всички страни на разработката.

За големи проекти, могат се използват разпределени екипи за да се намалят разходите, или поради специфични умения на локално ниво или за провеждане на специфичен локален маркетинг.

В разпределените екипи е особено важно да са добре определени начините за комуникация между отделните подекипи.

Трябва да се приложат стратегии за справяне със зависимости.

Методи за координация на артефактите – екипи, проекти

Възможностите за това са:

- неформалната комуникация;
- документацията;
- формалните срещи;
- електронната комуникация.

22.3. Прилагане

Ръководителят на екипа и архитектът взимат най-важните решения по време на тази фаза.

Тази фаза се разглежда от гледна точка на компромисите, разработката и проследяването на прогреса.

Компромиси (TradeOffs)

От гледна точка на ръководството компромисите са с времето, разходите, качеството, предоставената функционалност, краткосрочните и дългосрочните жалони. Всяко ново изискване се разглежда в рамките на време, цена и контекст на искането.

За всяка промяна се изяснява защо е необходима, влиянието й върху текуща версия, времето за реализацията и цената й.

Разработка

Разработката трябва да се извършва еволюционно. Всяка краткосрочна цел трябва да предоставя готова функционалност. Периодът на краткосрочните цели е от 2 до 6 седмици. Етапите за всеки период са планиране, разработка, тестване и отстраняване на грешки.

Етапи за всеки период:

- Планиране
- Разработване
- Тестване и отстраняване на грешки

Проследяване на прогреса

- Персонален контакт с разработчиците
  - Да се изяснява как върви разработката и има ли нужда от помощ.
  - В SCRUM, например, се провеждат ежедневни срещи.
  - Да не се извършва микроуправление при проследяването!!!
- Формални срещи за състоянието
  - В SCRUM спринтовете се извършва преглед на състоянието и се определят задачите
  - Срещите трябва да са ефективни – да не се забравя целта им!
- Метрики
  - Определят е значимите за проекта
  - Глобални метрики
  - Чисто софтуерни – редове код, брой класове ...
  - Брой свързани задачи
  - Отклонение от графика
  - Брой дефекти, които трябва да се фиксване
- Управление на риска

- Риск – потенциален проблем с негативни последствия
- Колкото по навреме се открие, толкова по евтино струва неговото разрешаване

22.4. Управление

Фокусът на управлението е върху външните сили влияещи върху проекта. Управлението се отнася за правилата за проследяване и контрол на софтуерните архитекти и техническите ръководители.

Има 4 основни точки за управлението:

1. Включване на система за контрол за всички важни решение на архитекта относно софтуерните елементи, дейностите и т.н.
2. Прилагане на система осигуряваща завършеност на изпълнението.
3. Създаване на процес за разработка и ефективното му управление.
4. Приложение и разработка на добри практики

Обзор

Всеки проект, както и софтуерният такъв трябва да е планиран, организиран, имплементиран, следен и правилно измерим

Има различни подходи за планиране – избираме най-подходящия за нашия контекст

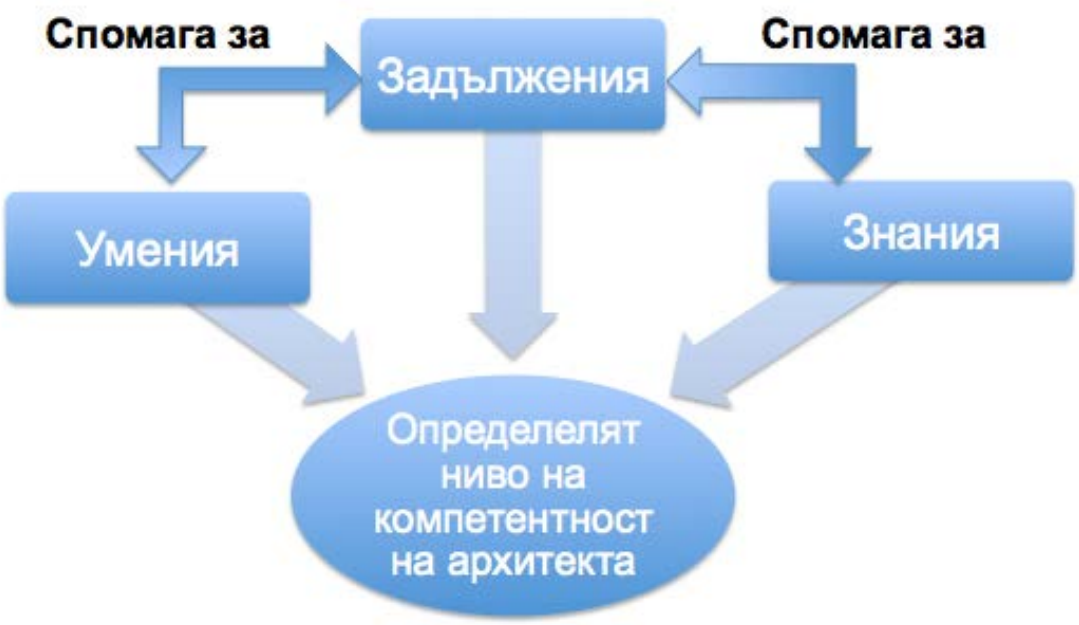
23. Модул 23: Компетентност на архитектурата

23.1. Компетентност на архитекта: задължения, умения и знания

Компетентност

Компетентността е набор от способности, знания, умения. Тя е основен източник на постигане на високи постижения в каквато и да е област. В случая ще разгледаме компетентността на архитекта, какво той е необходимо да знае, от какво да се интересува, за да можем да кажем, че софтуерът стоящ зад него е направен с високо качество и съобразен с най-вече с качествените атрибути, заложени зад този софтуер

Компетентност на архитекта: задължения, умения и знания



Уменията и знанията са основата на изпълняване на задълженията на служителите. Те от своя страна, работейки практически, покачват нивото на усъвършенстване на уменията и знанията му. Така трите лъча Умения, Знания, Задължения определят нивото на компетентност на служителя – в частност – архитекта

Компетентност на архитекта: задължения, умения и знания

Ще разгледаме различни примери от практиката на архитекта, с които се дава представа кои компоненти към кои от трите лъча на компетентността спадат

- **задължение**: “Дизайн на архитектурата” – като част от задълженията е архитектът да направи дизайн на архитектурата, според събраните функционални и нефункционални изисквания

- **умение** - “Свойство да мислиш абстрактно” – свойството на архитекта да мисли от стартова позиция “птичи поглед” и да създава по малки единици елементи и така рекурсивно до задоволителна и разбираемо ниво на абстракция

- **знание** - “Шаблони и тактики” – знание за дизайн и архитектурни шаблони, които би могъл да използва в практиката

Следващ такъв пример, основаващ се на трите компонента би могъл да е:

- **задължение** - “Документиране на архитектурата”
- **умение** - “Умение да пиша ясно”
- **знание** - “ISO Standard 42010”

Подобряване на индивидуалната архитектурна компетентност

- Ставай по-добър, изпълнявайки задълженията си. Учи се и практикувай.
- Подобрявай своите нетехнически умения в курсове за управление на хора и време.
- Обновявай знанията си за технологии, процеси, образци, тактики и т.н. чрез сертифициране, курсове, семинари и конференции.

Архитектът е не само техническо лице. Той е делегатор, основна връзка между различните заинтересовани лица, участващи косвено или не в проекта.

Задължения на архитекта

Основни елементи в длъжностната характеристика на един архитект в архитектурата и самият жизнен цикъл на проекта (life cycle) на проекта биха могли да бъдат различни в зависимост от контекста. В общия случай те са:

Архитектура:

- Създава архитектура – дизайн на софтуерната архитектура преди самото му имплементиране
- Оценява и анализира архитектурата – както по време на имплементация, така и по време на внедряване и изпълнения, с което се прави и оценка на софтуера
- Документира архитектурата – начин на представяне на архитектурата в различни изгледи за съответните заинтересовани лица
- Интегрира съществуващи системи – прилагане на пре-използваемост като качествен атрибут още на етап дизайн

Дейности в жизнения цикъл:

- Управление на изискванията – от самото начало на инициализиране на проекта до пускането му

- Имплеметиране на продукта/услугата – както писане на PoC, така и следне на имплементацията на проекта
- Тестване на продукта – създаване на основни тестови стратегии, участие в избора на технологи, с които ще се тества проекта
- Оценяване на нови технологии
- Избор на интрументи и технологии - взимане на избор на технологичният стек, в зависимост от знанията, уменията на останалите софтуерно свързани заинтересовани лица в проекта. Тук главна роля играят и заивисимостите които архитектите имат по отношение на бизнес ограниюенията като пари, време. Архитектът е в основата на избор на правилният избор на технологии

Задължения на архитекта

По отношение на управлението

- Управление на проекта
- Управление на хора
- Помага в Управлението / процеса

По отношение на организацията и бизнеса

- помага на организацията
- помага на бизнеса

Управлението на хора, добавянето на нови членове в екипа, в случай на нужда; разпределение на задачи и дейности в мениджмънта на проекта, е не само и единствено задължение на мениджъра на проекта. Архитектът е този, който знае технологичната част най-добре и той спомага за разпределението на хората с конкретни технологични знания, които най-добре биха се справили в имплементацията на проекта. Работи ръка за ръка с мениджъра в процеса на управление и организиране на хората, работещи по проекта

Умения на архитекта

Комуникационни умения

- Вътрешни за организацията – това как екипите в организацията комуникират, правила и стандарти. Интегриране на различни процеси за това
- Външни за организацията – как организацията комуникира с всички външни за нея трети лица: клиенти, страни доставчици и т.н.

Междуличностни умения

- Екипен играч / Team player
- С други хора – дипломатичност, респект

Работни умения

- Лидерство / Leadership
- Работа под напрежение
- Ефективност в приоритизацията

Архитектът е професионалист от към комуникация и начин на отношение към различни тип заинтересовани лица, свързани с проекта: вътрешни и външни за организацията. Той умее да обяснява и борави с комуникационни инструменти; умее да се справя с конфликти, ефективно да излиза от ситуации, с което да движи проекта напред

Умения на архитекта

Събиране / управление на информацията

- Да знае източници на информация
- Да вижда голямата картинка

Умения за справяне с неочаквани ситуации

- Допускане на двусмислие
- Взимане и менажиране на рискове

Знания на архитекта / Body of Knowledge

Архитектурни концепции

- Архитектурни рамки и шаблони
- Архитектурни шаблони
- Архитектурни тактики
- View points
- ADL’s
- Методи за оценяване на архитектурата
- Качествени атрибути

Знание на софтуерно инженерство

- Жизнен цикъл на софтуера (software life cycle)
- Системно инженерство
- Техники за подобряване
- Софтуерни продуктови линии

Знание за дизайн

- Различни инструменти и техники
- Дизайн на различни по тип системи – многонишкови, уеб сайтове и т.н.
- Обектно ориентиран дизайн, с UML и т.н.

Знание от програмни езици

- Различни по цел и контекст езици – функционални, обкетно ориентирани
- Може да се определят според случая
- Техники за realtime, сигурност, бързодействия и т.н.

Знание за технологии, рамки и шаблони

- Специфични рамки, шаблони и технологии
- Общо знание за рамки, шаблони и технологии

23.2. Компетентност на организацията

Компетентност на организацията

Организацията също си има своите задължения, умения и знания

Както архитектът, така и организацията може да се оцени от към гледна точка на ниво на компетентност. Разбира се, в този случай няма да се говори за личностно такова, а за набор от хора, процеси и приети практики в организацията

Пример би могъл да бъде:

- “намиране на добри архитекти” – **задължение**
- “провеждане на вътрешни курсове” – **знание**
- “Умение да се организират и провеждат курсове” - **умения**

Увеличаване компетентността на архитектите на организацията, чрез:

- Наемане на талантливи архитекти
- Създаване на кариерна пътека за архитекти
- Яснота по отношение на позицията на архитекта
- Яснота за задължение към архитекта
- Ментор на програма за архитекти
- Спомага за тренинги и курсове за архитекти
- Сертификации за архитектите
- Измерва продуктивността на архитектите
- Разработва пре-използваеми архитектури и артифакти

- Установи и затвърди ревю борд на архитектите
- Измерватели на качествтото на произведените продукти / архитектури

Въпроси по отношение на задължения, знания и умения

Има въпроси, казуси по отношение на трите стълба – задължения, знания, умения - до които всяка организация стига:

- Как се създава една архитектура?
- Как се анализира и оценява архитектурата?
- Как организацията осигурява адекватни знания на архитектите?
- Как се проектира архитектурата и как се разпределя работата по екипи ?
- Как се следи за цената ?
- Как се събира и споделя знанието ?

Обзор

Знанията и компетентността на един архитект са не само технически.

Трите основни стълба, както за организационна, така и за архитектурна компетентност са знания, умения и задължения. Те живеят и се развиват заедно.

24. Модул 24 Архитектура и софтуерни продуктови линии

24.1. Софтуерна продуктова линия

Софтуерна продуктова линия

Софтуерната продуктова линия описва софтуер, който се позовава на два главни компонента: общ и специфичен такъв. Общият софтуерен компонент се използва за всички клиенти на продукта, докато специфичният е предназначен за конкретен клиент или набор от клиенти. Това разделение спомага за бързото и ефективно разработва на такъв продукт, наименуван софтуерна продуктова линия.

Едни от двигателите за софтуерната архитектура са пестенето на време и пари

Стратегии като:

- Прилагане на шаблони, с които не се налага преоткриване на “топлата вода”, за новата версия на продуктовата линия
- Прилагане на стилове – по-високо ниво на абстракция от шаблоните, с които се създават правила и дизайн решения на архитектурно ниво и по високо ниво, не на модулно такова както при шаблоните за дизайн
- Интеграция на готови рамки (framework)

Неимоверно допринасят за тази цел, като във всичко това лежи принципа на повторната използваемост като основна тактика.

Софтуерна продуктова линия – цел

- Олицетворение на стратегическо, планирано пре-изпозлване
- Много повече от просто нова технология
- Нов начин на правене на бизнес

- Основни двигатели: **време**, **пари** и **качество**

Софтуерната продуктова линия – същност

- Набор от софтуерни системи

Делящи **общи**, менажиращи се характеристики

Задоволяващи **специфичен** маркетингов сегмент

Разработвани от общ набор от **активи/софтуерни единици** в предопределен вид/план

Софтуерна продуктова линия – рамка/шаблон/framework

Трите лъча при произвеждането на продуктовата линия са:

- Мениджмънт, това как се менажира такъв вид продукт. Има специфики, на които е необходимо да се наблегне, като избор на процес, взимане на решения
- Основна (Core) софтуерна разработка: основните елементи, които са част от коя да е версия на софтуерната продуктова линия
- Продуктово разработване – специфичните софтуерни елементи за съответната версия продуктова софтуерна линия, базата за които са core елементите

Софтуерна продуктова линия
защо ?

На базата сумарни проучвания на различни набори от софтуерни продукти се доказва ефективното прилагане на продуктовите линии. Една такава сумарна статистика е:

- Увеличава продуктивността с до 10 пъти
- Увеличава качеството с до 10 пъти
- Намалява цената с до 60%
- Намалява нуждата от адвокати с до 87%
- Намалява времето на маркетингово проучване с до 98%
- Възможност да влезнеш на пазара с месеци, не с години

Пример на софтуерна продуктова линия
проблем - Софтуер за банково кредитиране

Ще разгледаме пример на софтуерна продуктова линия в банковата сфера. Ще искаме финансов софтуер, който има както общи за банковите институции услуги, така и специфични за всяка такава.

Сумарно изискванията са:

- Работещ за 20 конкретни банкови институции
- Подобни имплементации за всички банки (клиенти на системата)
- Всяка има специфични за нея, под стъпки
- Как един архитект и софтуерен екип би задоволил всички банки ?
- “Вмъквайки” конкретен код за всяка банка във целия такъв ?

Пример на софтуерна продуктова линия
решение – Variability

фиг. 24.2.1

На базата на core assets, които са основните услуги и функционалности, от които се черпи за отделните версии на продуктовата линия, се създават самите те – версиите й. Всяка версия може да избира определен набор от функционалности от core assets или всички такива с различни конфигурационни файлове, примерно, които да задават кои части или версии от софтуера да се използва. С това всеки клиент може да избира съответна версия от продуктовата линия

24.2. Какво прави една Софтуерна продуктова линия работеща?

Какво прави една Софтуерна продуктова линия работеща?

Изисквания

- Базови – core assets
- Специфични – “delta” документи

Архитектурен дизайн

- Залагане на най-вече modifiability, reliability и подобни качествени атрибути най-вече

Софтуерни елементи

- Крос домейн
- Преизползваем софтуер
- Inversion of Control

- Contract base софтуер

Какво прави една Софтуерна продуктова линия работеща?

Моделиране и анализ:

- модели на изпълнението;
- разпределени системи;
- Интернет политики.

Тестване

- процедури на тестване;
- системи за тестване.

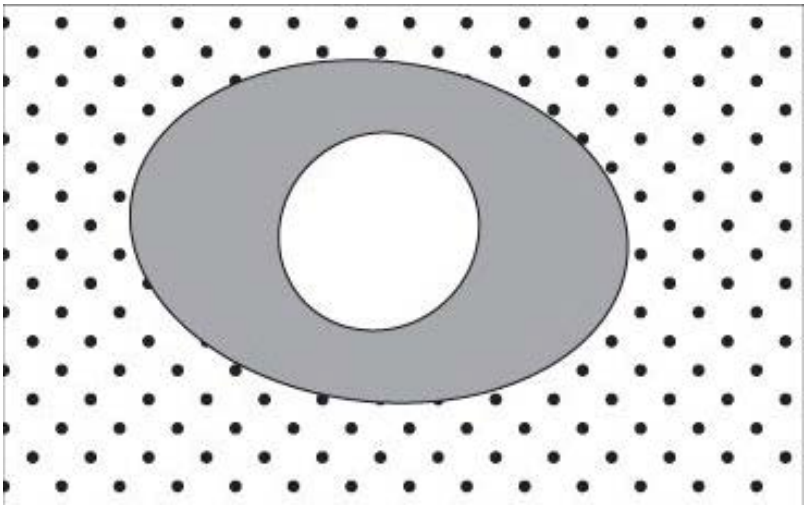
Планиране на проекта: бюджет, план, разпределение на работата.

24.3. Обхват на продуктовата линия

Обхват на продуктовата линия

Примерно разпределение на системи по отношение на обхват

- С предопределим обхват (бяло)
- Извън обхвата (на петна)
- С нуждаещ се от постъпкови решения (сиво)



фиг. 24.4.1.

Всяка продуктова линия има ниво на неопределеност на обхвата от към време на работа, необходими средства и ресурси.

На фиг. 24.4.1 се дава съотношение между различните нива на неопределеност – предопределим, извън обхвата, нуждаещ се от постъпкови решения

С предопределен обхват

- Добре дефинирано ядро от активи.
- Добри приложими тактики за постигане на атрибутите за качество.
- Лесно определяне на компонентите базирани на ядрото от активи.
- Прогнозируем софтуер.

Извън обхвата, нуждаещ се постъпкови решения

- Необмислено/недефинирано ядро от активи.
- Проблеми при дефинирането на обхвата - мерни единици – време, пари, средства?
- Разработката в движение е враг!

24.4. Атрибут за качество вариантност

- Асоцииран е с продуктовите линии.
- Задоволява общото и вариациите, идентифицирани от обхвата на продуктовата линия.
- Специална форма на изменяемост.
- Ядрото от активи се адаптира в различните контексти на продуктовата линия.
- Целта е разработка на лесни продукти/услуги в концепцията на продуктовата линия.

Сценарии

Източникът на стимула изисква вариантността.

Стимулът е да се поддържат вариации на хардуер, изисквания, технологии, потребителски интерфейси, атрибути за качество и т.н.

Среда по време на изпълнение, изграждане, разработка.

Артефакти: изисквания, архитектура, компоненти, план на проекта.

Отговор: вариациите/изменяемостта може да се създаде/проектира.

Замерване на отговора: време и пари за да се създаде вариация с използване на ядрото от активи.

24.5. Ролята на архитектурата в Продуктовата линия

Тактическа:

- Ясно дефиниране на това, какво се очаква.
- Яснота за това, какво е в наличност.
- Софтуер идеализиран за вариации.

Стратегическа:

- Евтина доставка на софтуер.
- Бърза доставка на софтуер.
- Ефективна доставка на софтуер.
- Конкурентоспособност на пазара.

24.6. Механизми за вариации

- Добавяне/премахване на елементи;
- Добавяне на променящи се елементи;
- Избор на различни версии на елементи:
  - с един интерфейс;
  - с различно поведение;
  - с различни атрибути за качество.
- Точки на разширяемост - места, където може да се добави поведение.
- Рефлективност - манипулиране на данни чрез метаданни.
- Презареждане – чрез промяна на поведение според типа.

24.7. Оценяване архитектурата на продуктовата линия

Оценяване архитектурата на продуктовата линия

Какво и как да се оценява?!

- Фокус по конкретни точки на вариации
- Бързина на създаване
- Гъвкавост

Кога да се оценява?! - Когато продукта следва да бъде различен за различни клиенти

24.8. Ключови въпроси относно продуктовата линия

Стратегии на адаптивност

Отгоре-надолу спрямо отдолу-нагоре:

- зависи от политиката в организацията;
- избор според контекста.

Проактивна спрямо реактивна:

- проактивна – бързо взимащи се стратегически решения;
- реактивна – продукти направени от по-ранни такива.

С нарастване спрямо Големия взрив:

- с нарастване – постъпково, евалюционно, поетапно;
- Големият взрив – всичко наведнъж.

Създаване на продукти и оценка на Продуктовите линии

Оценяване водено от:

- Външни източници
  - нови версии на софтуерни елементи;
  - нови софтуерни елементи, кандидати да бъдат добавени/интегрирани;
  - нови характеристики.
- Вътрешни източници
  - нови функции на продуктите;
  - смяна на продуктите.

25. Модул 25 Архитектури в облака

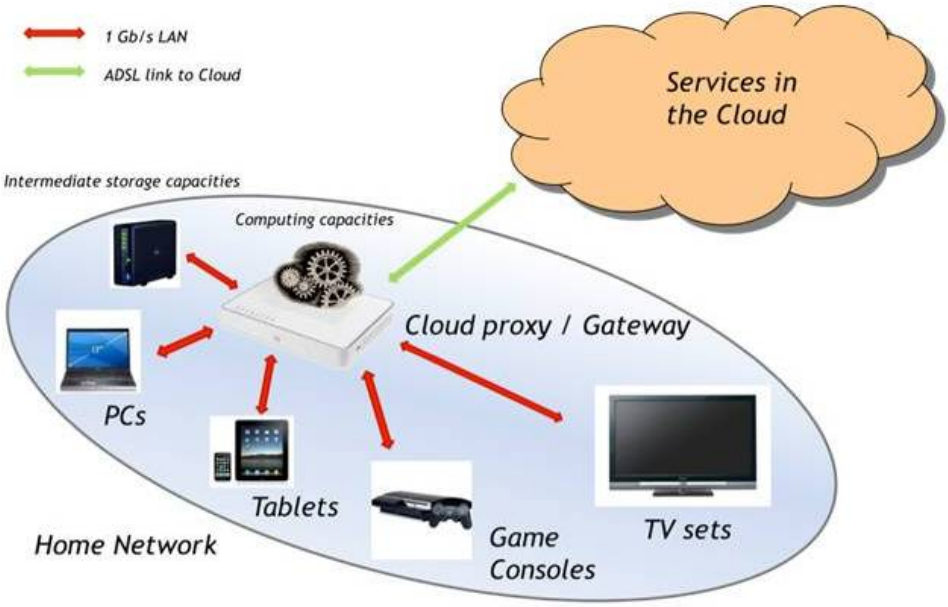
25.1. Облак (Cloud)

В този модул се разглежда съвременно инфраструктурно решение, наречено облак или cloud. То дава възможност на клиента, нуждаещ се от услуги да плаща само за нужните му количества като бройка регистрирани потребители, процесорна памет, хардуерни мощности и т.н.

Cloud – Облак

Начин на предоставяне на услугите:

- свободно добавяне/премахване на ресурси;
- мащабиране според нуждите;
- заплащане според използването.



фиг. 25.1.1.

На фиг. 25.1.1. се дава представа на това как работи облака. Услугите или още – облачните услуги виреят на различни сървъри, като те могат да бъдат в коя да е точка на земята. Има load balancing, който се грижи за връзка между заявка от кой да е тип клиентско устройство и търсена услуга от облака

25.2. Основни определения в облака

Основни определения в облака

Самообслужване при нужда(On – demand self-service): Автоматично добавяне на нови ресурси: дискове, процесори, памет, виртуални машини и т.н., без намесата на оператор.

Повсеместен мрежови достъп (Ubiquitous network access**):**

- достъп от интернет чрез браузър;
- достъп от мобилни телефони, лаптопи, таблети и т.н.;
- независим достъп от мястото и устройството;
- лесен начин на управление и анализ.

Споделяне на ресурси / Resource pooling

- Заложено е ползване на общи (pooling) компютърни ресурси
- Обслужват се множество потребители
- Динамично разпределяне на вертикални или хоризонтални ресурси на потребителите
- Location Independency (Независимост от локацията)
- Дефинира се абстрактна локация

Measured service (измерима услуга)

- Всеки ресурс се използва с измерима единица
- Контрол на процента заетост
- Оптимизация на заетостта на ресурсите
- Следена и управление на заетостта на ресурсите

Много наематели/Multi-tenancy

- Едно приложение поддържа множество “класове” от потребители
- Клас потребител има своя база данни, права на достъп
- Различни класове потребители се менажират и дистанцират от приложението

25.3. Модели услуги и начини на използване

Модели услуги

Софтуера като услуга (Software as a Service - SaaS): Консуматор на услугата е крайният потребител. Приложенията са разположени в облака – приложения в реално време, потоци от видео, календари и т.н. Потребителят не контролира софтуера и инфраструктурата.

Платформа като услуга (Platform as a Service - PaaS): Консуматор на услугата е разработчика на софтуер. Сред услугите са системи за управление на бази от данни, балансиране на натоварването, управление на инфраструктурата и т.н. Атрибути на качеството са време за отговор, сигурност, възстановяване от провал.

Инфраструктурата като услуга (Infrastructure As a Service - IaaS): Консуматорът на тази услуга е разработчика на софтуер или системния администратор. Позволява управление на инфраструктурата, използването на процесорите, виртуалните машини и т.н.

Модели услуги - примери

Начини на разгръщане

Частен облак (private cloud):

- Облакът е собственост на една организация;
- В облака се изпълняват само приложенията на организацията, която го притежава.
- Обикновено, приложенията са оперативните системи на организацията.

Публичен облак (public cloud):

- Има публичен достъп към облака.
- Облакът е собственост на една организация.

Обществен облак (community cloud):

- Общ за няколко организации. Поддържа специфична общност с общи интереси: мисия, изисквания, политика.

Хибриден облак (hybrid cloud):

- Композиция от два или повече облака (частен, публичен или обществен).

25.4. Икономическа ефективност

Икономии от мащабиране нагоре (scale upping)

Големи центрове за данни са по-скъпи за поддръжка от малките поради големия брой сървъри и по-скъпите операции. Факторите на цената са:

- Цена на електричеството: 15-20% от цялата цена на услугата е мощността, използвана за облака.
- Цена на труда: В големите центрове за данни се автоматизират операциите. Един администратор управлява хиляди сървъри в облака.
- Сигурност и надеждност: В облака се управлява нивото на сигурност по-евтино отколкото за отделен сървър. Бързо се възстановява от провали.

Икономии от мащабиране нагоре (scale upping)

Фактори на цената

- Хардуерни цени
- Поддръжката на инфраструктури в облачната архитектура излиза по евтино от тези за отделни сървъри
- Намаления за големи количества

Начини на изпозлване в облака

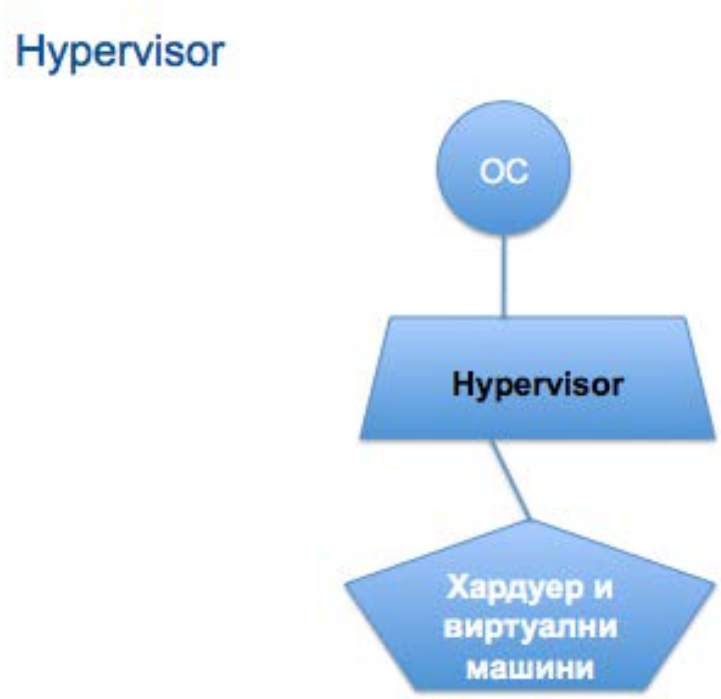
Използване на виртуални инфраструктури. Източници на вариация:

- Произволен достъп
- Време от деня
- Време в годината
- Използване на шаблони за ресурсите
- Децентрализиране

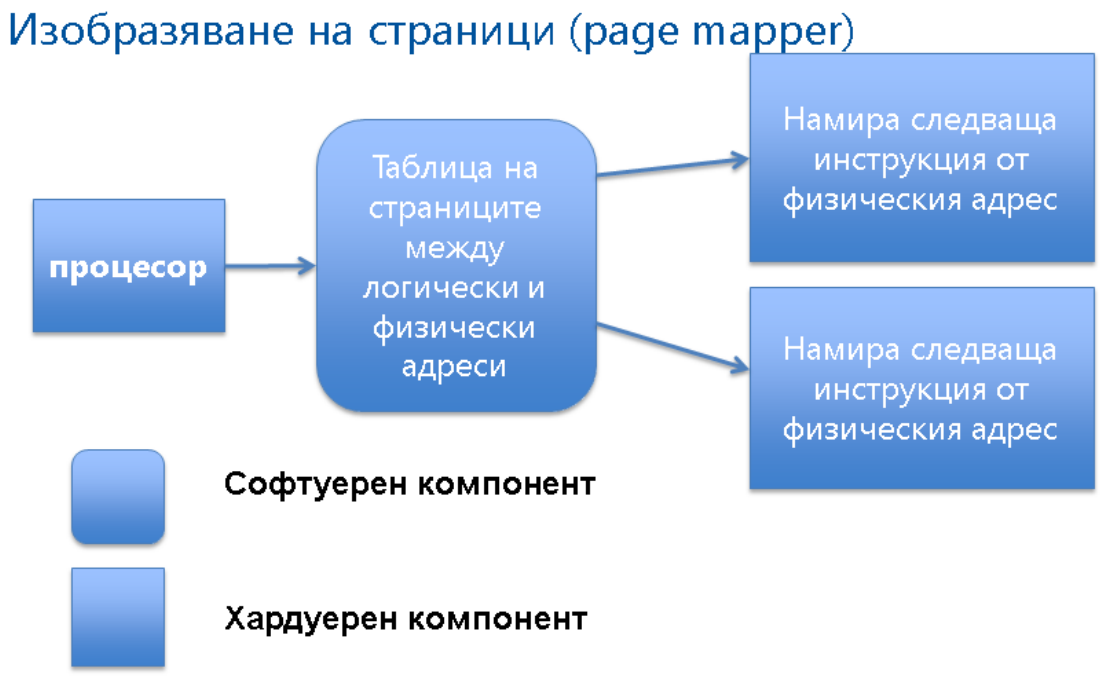
Много наематели / Multi-tenancy

Едно и също приложение се използва от различни групи/класове от потребители. Това позволява намаление на цената при обновяване или управление. Всяко обновяване се извършва на един екземпляр за всички потребители, а не върху множество от сървъри.

25.5. Базови механизми на предоставяне на услуги



При този тип предоставяне на услуги виртуалните машини заемат основни място. Те са основно средство за скалируемост и поддръжка.



При page mapper облака имаме таблица от услуги и физическо хостване на тези услуги. През тази таблица минава коя да е заявка, преди да бъде обработена

Съхранение на данни (Storage)

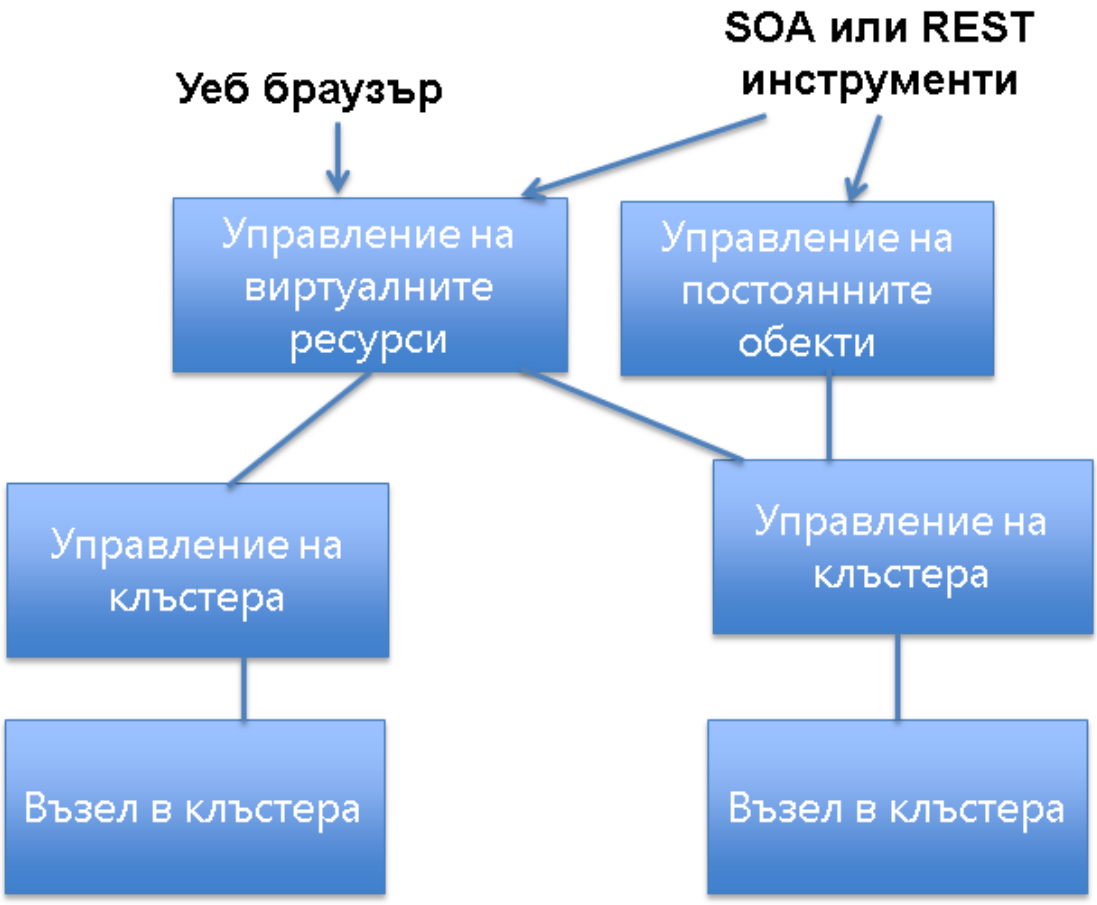
- Менажиран от различни физически сървъри
- Клъстър базиран
- Hadoop Distributed file System е примерна такава услуга

25.6. Примерни технологии

Примерни технологии в IaaS

- Три основни услуги на виртуализацията
  - Виртуализация на изчисленията
  - Виртуализация на мрежата
  - Виртуализация на файловата система

IaaS гледка на разпределението (IaaS allocation view)



Получава се заявка на виртуалният мениджър на ресурсите, която от своя страна определя клъстера, на който реално е разположен изисквания софтуер. Инструменти за имплементация могат да са SOA, REST архитектурните стилове

IaaS гледка на разпределението (IaaS allocation view)

- **Управление на виртуалните ресурси(**Virtual resource менижър): предоставя виртуален достъп до определен клъстер.
- **Управление на клъстера (**Cluster мениджър): отговаря за управлението на ресурсите в клъстера.
- **Управление на постоянните обекти (**Persistence Object Manager): поддържа управлението на моделите на приложната област.

Примерни технологии в PaaS

- Предназначен е за разработка и разгръщане в облака.
- Предоставя платформа на софтуерните инженери в облака: технологии, готови приложения, хардуер.
- Примерни доставчици: Google App Engine, Microsoft Azure Engine.

Примерни доставчици

- Google App Engine
- Microsoft Azure Engine

Примерни технологии в базата данни

- Релационни база данни като MySQL, Oracle с възможности за клъстеризация.
- Не релационни бази данни като MongoDB, Hbase, Neo4J.

Това каква система за управление на бази от данни ще се използва зависи най-вече от целта и контекста на приложението.

Архитектури в облака

Този различен по тип и концепция архитектура предразполага за по различен начин за разглеждане на някои качествени атрибути, като

- Сигурност/Security
- Производителност / Performance
- Разполагаемост / Availability

Архитектури в облака – аспекти на сигурността

Технически и нетехнически аспекти

Не технически

- Местни за доставчика на облака, разпоредби и закони
- Физическа сигурност, която се предоставя от доставчика

- Вътрешно-организационна сигурност

o Технически - предпазване от различни видове атаки, като атаки по страничните канали, отказ от услуги, избягване на виртуалната машина.

Архитектури в облака – производителност (performance)

- Заделяне на ресурси според нуждите
- Scale up – машабиране нагоре, където се пускат повече хардуерни мощности
- Клъстеризация
- Разпределение на натоварването (Load balancing) – механизъм за по бързо обслужване, чрез разпределяне на заявките между няколко сървъра

Архитектури в облака – разполагаемост (availability)

Клъстеризацията спомага за постигане на "големите" 9-ки, т.к. се разпределя натовареността на заявките. Дори в случай на падане на отделен сървър, то заявките се обработват от друг сървър на облака.

Общоприета надежност: 99,99 % (4 деветки/nines).

Обзор

- Cloud е алтернатива при хостване на услуги
- Огромен плюс е начина на плащане – плащаш, това което използваш.
- Времето за внедряване на нови услуги е много кратно по-малко, поради елиминиране на доставките на нов хардуер и съкращаване на времето за осигуряване на базови услуги, като инсталация на операционни системи и др.

26. Списък с полезни препратки

- Software Architecture in Practice, 3-th edition, Len Bass
- http://agilemethodology.org/ обяснява в детайли agile методология и конкретни имплементации като SCRUM и Kanban
- http://iasaglobal.org/ световна организация на сотуерните архитекти
- http://www.opengroup.org/ световна организация за IT стандарти – архитектури, процеси, технологии
- Patterns of Enterprise Application Architecture, Martin Fowler – набор от шаблони за бизнес софтуерни приложения и системи
- Design Patterns, Elements of Reusable Object-Oriented Software, Erich Gamma – описва основните софтуерни шаблони за програмиране
- Ориентираната към услуги архитектура за бизнеса, В.ладимир Димитров , УИ "Св. Климент Охридски", 2012