ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Катедра: „Информатика и софтуерни науки”
Дисциплина: „Софтуерни архитектури”

Р Е Ф Е Р А Т

Тема: „Софтуерни архитектури”

Десислава Емилова Милушева
ИСН, курс II, гр. 77,
Фак. № 471219007

Разработил: …….
/ Десислава Милушева/

Проверил: ……
/ ас. Виктор Главев /

София, 2021г.

Съдържание

1. Увод

Светът става все по-зависим от софтуера. Софтуерът е съществен елемент от всеки мобилен телефон, лаптоп, IoT смарт устройство, автомобил и дори от сложните системи за контрол на въздушното движение. Всъщност много от иновациите, които сега приемаме за даденост - включително бизнес моделите на големи компании за търговия на дребно като eBay или Amazon - просто нямаше да съществуват, ако не беше софтуерът. Традиционни правителствени и държавни организации, като тези, които се намират във финансовия, търговския и публичния сектор, силно зависят от софтуера. В наши дни е трудно да се намери организация или предприятие, които по някакъв начин да не са в софтуерния бизнес или да не са много силно и пряко зависими от софтуерните модели, които ги движат.

За да оцелеят такива компании, организации и дори малки предприятия, софтуерът, от който зависят, трябва да предоставя необходимата функционалност и способност, да бъде с достатъчно високо качество и добре разработен модел, да е наличен и възможен за лесно интегриране и внедряване, когато е обещан в срок, да е лесно поддържан, и да се доставя на приемлива цена. Всички тези характеристики са силно повлияни и пряко зависят от архитектурата на софтуера.

Архитектурата на системата описва основните й компоненти, техните взаимоотношения (структури) и как те взаимодействат помежду си. Ето защо е изключително важно планирането, структурирането и развитието на всеки едно софтуерен проект да стартира с фундаментално, първично, стартово отделено време, за идейно, базово разработване и изясняване на софтуерната архитектура, защото това е основата на всички успешни проекти.

Разгръщайки темата за същността и имплементацията на софтуерната архитектура, в следващите подтеми ще бъдат разгледани някой основни принципи и важни твърдения, които лежат в основата на изграждането на една добре издържана, оптимално работеща и стабилна софтуерна система. Ще бъде дефинирана идеята внедрена в понятието софтуерна архитектура.

2. Какво представлява софтуерната архитектура? Дефиниции

Софтуерната архитектура може да бъде дефинирана като определянето и структурирането на решение, което отговаря на техническите и оперативните изисквания нужни за създаването на софтуер. Софтуерната архитектура оптимизира атрибути, включващи поредица от решения, като сигурност, производителност и

управляемост. Тези решения в крайна сметка оказват влияние върху качеството на приложението, поддръжката, производителността и общия успех.

В основата на цялата дискусия за софтуерната архитектура е фокусът върху разсъжденията относно структурните проблеми на системата. И въпреки че понякога архитектурата се използва за означаване на определен архитектурен стил, като клиент-сървър, а понякога се използва за обозначаване на област на изследване, тя най-често се използва за описване на структурни аспекти на определена система. Тези структурни проблеми са свързани с разработването софтуерната архитектура, която преди всичко може да се каже, че е форма на софтуерен дизайн, който се появява най-рано при създаването на системата и въвежда абстрактно ниво за начина на работа.

Според книгата „Софтуерна архитектура на практика" съвместно със софтуерния инженерен институт „Carnegie Mellon", нейните автори, Лен Бас, Пауло Клементс и Рик Казманцернс, предлагат следната дефиниция:

„Софтуерната архитектура на системата представлява набор от структури, необходими за изграждането на системата, които включват софтуерни елементи, връзките между тях и свойствата на двете." [1]

Структурните въпроси или проблеми около софтуерната архитектура на една система обикновено включват примерна или образна организация съвместно с глобална структура за контрол на работата. Важни аспекти от реализацията са свързани с въпросите относно протоколи за комуникация, синхронизация и достъпа до данни, роля и основи на функционалност и действието на дизайнерските елементи, физическо разпределение, композиция на дизайнерски елементи, мащабиране и производителност и не на последно място избор между алтернативи на самия дизайна.

Все пак дадената дефиницията по-горе не може да бъде определена като крайно конкретна и различни научни трудове биха описали различни определения за софтуерната архитектура. Общото би било подчертаването на няколко от основните структурни проблеми и съответните начини за тяхното описание. Всеки от тези проблеми може да бъде проследен до идеята му и това, което софтуерния разработчик цели да разреши в основата на създаването на софтуерната архитектура - да я анализира, да я развие, да я представи или да се развие от нея.

Макар да изглежда объркващо да има множество интерпретации, тези различни интерпретации не се изключват взаимно, нито представляват основен конфликт относно това какво представлява софтуерната архитектура. Те заедно

[1] Software Architecture In Practice / Len Bass, Paul Clements, Rick Kazmancerns. - 3rd ed., 2013 (Software Engineering Institute) - p.4

представляват спектър в общността за изследване на софтуерната архитектура относно акцента, който трябва да се постави върху архитектурата - нейните съставни части, цялото обект, начина, по който се държи веднъж построен софтуер, или надграждането над него. Взети заедно, всички дефиниции формират консенсусно мнение за софтуерната архитектура и дават по-пълна картина.

Фундаментално, софтуерната архитектура е свързана със структурните свойства на една системата. Структурните свойства могат да бъдат изразени чрез компоненти, взаимовръзки и принципи/насоки на тяхното използване. Точните структурни свойства, които трябва да се разгледат, и начините на тяхното представяне варират в зависимост от това, което представлява структурен интерес за потребителя на архитектурата.

3. Защо софтуерната архитектура е важна?

Софтуерната архитектура е в основата на всяка една софтуерна система. Подобно на други видове инженерство, това, което се намира в основата, има силен ефект върху качеството на изграденото върху нея. Като такава, софтуерната система има голямо значение по отношение на успешното развитие и евентуална поддръжка на дадена система.

Софтуерната архитектура е поредица от решения. Някои от най-ранните решения идват от проектирането на архитектурата и те имат висока степен на важност, защото влияят върху решенията, които идват след нея. Самата концепция е свързана с много от решенията на корпоративната организация, която изгражда дадено софтуерно решени и би се отразило не само на качеството на продукта, но и на финансовата страна относно развитието на тази компания и бизнесите, които тя обслужва. Тук изключително важна роля играе не само самата архитектура, а и екипа или човека, които преценяват необходимостта. Според Марк Ричърдс и Неал Форд, архитекта на един софтуер е натоварен с нелека задача, защото:

„От архитектът се очаква да определи архитектурните решения и принципите на проектиране, използвани за насочване на технологичните решения в екипа, отдела или в предприятието.” [2]

Друга причина софтуерната архитектура да е важна е, че всички софтуерни системи имат архитектура. Дори да се състои само от една структура с един елемент, има архитектура. Има софтуерни системи, които нямат официален дизайн, а други, които официално не документират архитектурата си, но дори тези системи все още имат архитектура. Колкото по-големи са размерите и сложността на софтуерна

[2] Fundamentals of Software Architecture / Mark Richards, Neal Ford. - 1st ed., 2020 - p.7

система, толкова повече ще е необходима добре обмислена архитектура и планирана архитектура, за да бъде изграден успешно функциониращ софтуер.

Софтуерната архитектура предоставя редица предимства, когато се прави правилно, което значително увеличава шансовете за успех на софтуерната система. Правилната основа, заложена от архитектурата на софтуерната система, дава редица предимства, които водят до отговора на въпроса - защо софтуерната архитектура е важна.

Както всяка друга сложна структура, софтуерът трябва да бъде изграден на здрава основа. Ако не бъдат разгледани ключови сценарии, на може да бъде проектирана успешна система, защото биха се появили често срещани проблеми, които от своя страна биха довели до дългосрочните последици. Липсата на стабилни решения, можете да изложите приложението на риск и провал. Съвременните инструменти и платформи помагат да се опрости задачата за изграждане на софтуери, но те не заместват необходимостта от внимателно проектиране, въз основа на специфичните сценарии и изисквания към конкретният продукт, които ще бъде създаден. Рисковете, породени от лоша архитектура, ще доведат до софтуер, който е нестабилен, не е в състояние да поддържа съществуващите или бъдещите бизнес изисквания или е труден за внедряване или управление в производствена среда. Системите трябва да бъдат проектирани с отчитане на потребителските очаквания, заобикалящата ИТ инфраструктура и бизнес целите на клиента. За всяка от тези области трябва да се очертаят ключови сценарии и да се идентифицират важни атрибути на качеството (например надеждност или мащабируемост).

Могат да бъдат отделени три основни причини за важността на архитектурата изграждаща стабилната платформа върху, която един софтуер може да са развива. На първо място, софтуерната архитектура играе ролята на свързваща единица между компонентите, което улеснява взаимното им “общуване”. Тя представлява обща абстракция на високо ниво, която позволява всички страни в една система да я използват като основа за създаване на взаимно разбирателство, формиране на консенсус и комуникация помежду си.

На второ място софтуерната архитектура представлява въплъщение на най-ранния набор от дизайнерски решения за дадена система. Тези ранни обвързвания имат тежест далеч непропорционална на индивидуалната им тежест по отношение на оставащото развитие на системата, нейната услуга в разгръщане и нейния живот на поддръжка в бъдеще.

На последно място, но не и по важност софтуерната архитектура прехвърля абстракцията на една система и въплъщава сравнително малък, интелектуално-разбираем модел за това как е структуриран софтуера и как всички компоненти работят заедно. Този модел може да се прехвърля в различни системи или по точно може да приложим рекурсивно неговото действие и към други системи, показващи подобни изисквания. Така би се създала една картина на повторно преизползване на един вече работещ модел с цел по-голям мащаб на дадена система.

4. Какво е архитектурен шаблон? Видове архитектурни шаблони.

Софтуерните архитектурни шаблони за проектиране са от решаващо значение при решаването на комплексни проблеми при разработката на индустриални програмни приложения. Създаването на софтуер в съвременността безспорно се свърза с бързина, качество, ефективност, преизползваемост и други не-функционални качествени характеристики, които се явяват като предусловия за започването на какъвто и да е софтуерен проект. Това предсказва основата на имплементирането на вече готови или полуготови решения, представящи се най-често с дефиницията на шаблон.

Шаблонът е образец на софтуерно решение на проблем в определен контекст и домейн. Той е от решаващо значение при имплементирането на комплексни казуси при разработката на индустриални програмни приложения. Шаблоните предлагат елегантни решения на типови проблеми в проектирането с възможност за многократно използване. С тях може да се структурира изцяло темплейт-ориентирана софтуерна разработка. Според авторите на книгата „ Design Patterns: Elements of Reusable Object-Oriented Software Patterns relationships" може да бъде дадено следното определение:

„ Софтуерен шаблон за дизайн представлява концепция за разрешаването на често срещан проблем в софтуерната разработка в конкретен контекст и домейн." [3]

Могат да бъдат разграничени пет основни архитектурни шаблона:

- Многослоен шаблонен модел
- Шаблонен модел на микроядрото (шаблон на приставката)
- Шаблон CQRS (Сегрегация на отговорността за командване и отговорност)
- Шаблонен модел на източници на събития
- Шаблонен модел на микроуслугите

Многослойният модел е може би един от най-известните модели на софтуерна архитектура. Идеята му е кодът да се раздели на „слоеве", където всеки слой носи определена отговорност и предоставя услугата на по-висок слой. Няма предварително определен брой слоеве, но най-често срещани са:

1. Слой на потребителския интерфейс
2. Приложен слой

[3] Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnsonq John Vlissudes . - November, 1994 – p.13

3. Бизнес или домейн слой
4. Слой за достъп до данни
5. База данни

Идеята е потребителят да инициира част от кода в слоя на потребителския интерфейс, като извърши някакво действие (например щракване върху бутон). След това потребителския слой извиква основния слой, т.е. приложния слой. Последва влизане в бизнес слоя и накрая, слоя за достъп до базата данни съхранява всичко в самата базата данни. Така че по-високите слоеве зависят от и извършват повиквания към по-ниските слоеве. Някои приложения могат да пропуснат слой, възможно е дори да се обединят два слоя в един. Всеки слой има своя собствена отговорност. Слоят на потребителския интерфейс съдържа графичния дизайн на приложението, както и всеки код за обработка на потребителското взаимодействие. Бизнес слоят е мястото, където се поставят моделите и логиката, които са специфични за бизнес проблема, който трябва да се разреши. Приложният слой осигурява абстракция, така че слоят на потребителския интерфейс да няма нужда да познава бизнес слоя. И накрая, слоят за достъп до данните съдържа кода за достъп до слоя на базата данни. Слоят на базата данни е основната технология на базата данни (например SQL Server, MongoDB). Този модел осигурява лесен начин за писане на добре организирано и лесно за тестване приложение.

Шаблонният моделът на микроядрото или шаблон на приставката е полезен, когато приложението има основен набор от отговорности и колекция от взаимозаменяеми части. Микроядрото ще осигури входната точка и общия поток на приложението, без наистина да знае какво правят различните приставки. Микроядрото може да съдържа цялата логика за планиране и задействане на задачи, докато приставките съдържат специфични задачи. Докато приставките се придържат към предварително дефиниран API, микроядрото може да ги задейства, без да е необходимо да знае подробности за изпълнението. Този модел осигурява голяма гъвкавост и разтегливост. Някои реализации позволяват добавяне на приставки, докато приложението работи. Микроядрото и приставките могат да бъдат разработени от отделни екипи. Недостатъците на този модел са, че може да бъде трудно да се реши кое принадлежи в микроядрото и кое не.

Концепцията на шаблонният модел за сегрегация на отговорността за командване и отговорност (CQRS) е, че приложението има операции за четене и операции за запис, които трябва да бъдат напълно разделени. Това също означава, че моделът, използван за операции по запис (команди), ще се различава от моделите за четене (заявки). Освен това данните ще се съхраняват на различни места. В релационна база данни това означава, че ще има таблици за командния модел и таблици за прочетения модел. Някои реализации дори съхраняват различните модели в напълно различни бази данни, напр. SQL Server за командния модел и MongoDB за прочетения модел. Когато потребителят извърши действие, приложението изпраща команда до командната услуга. Командната услуга извлича всички необходими данни от командната база данни, прави необходимите манипулации и ги съхранява обратно в базата данни. След това уведомява услугата за четене, за да може моделът за четене да бъде актуализиран. Когато приложението

трябва да покаже данни на потребителя, то може да извлече модела за четене, като извика услугата за четене. Предимството на този шаблон е, че командните модели могат да се фокусират върху бизнес логиката и валидирането, докато моделите за четене могат да бъдат съобразени с конкретни сценарии. Можете да се избегнат сложни заявки (напр. Обединения в SQL), което прави четенията по-ефективни. Недостатъкът е, че синхронизирането на командата и четените модели може да стане сложно.

CQRS моделът често върви ръка за ръка с шаблонния модел на източниците на събития. Това е модел, при който не се съхранява текущото състояние на вашия модел в базата данни, а по-скоро събитията, които са се случили с модела. Така че, когато името на клиент се промени, няма да се съхранява стойността в колона „Име“. Ще се съхрани събитие “ПромянаНаИме” с новата стойност (и евентуално и старата). Когато трябва да се извлече модел, извличат се всички съхранени събития и се прилагате отново върху нов обект. Това е вид преизползване на обект. Аналогия в реалния живот на този шаблон може да бъде направена със счетоводството. Когато се добави разход, не се променя стойността на общата сума. В счетоводството се добавя нов ред с операцията, която трябва да се извърши. Ако е допусната грешка, просто се добавя нов ред. За да улесните живота си, можете да изчислите общия брой всеки път, когато добавите ред. По подобен начин работи и шаблонния модел на източникът на събития, никога не премахва събития, защото те несъмнено са се случвали в миналото, а вместо това, за да се коригират ситуациите се добавят нови събития.

Шаблонният модел на микроуслугите разглежда системата като набор от микроуслуги, или множество приложения, които ще работят заедно. Всяка микроуслуга носи своя отделна отговорност и екипите могат да ги разработват независимо от другите микроуслуги. Единствената зависимост между тях е комуникацията. Тъй като микроуслугите комуникират помежду си, ще трябва да се уверите, че съобщенията, изпратени между тях, остават обратно съвместими. Това изисква известна координация, особено когато различни екипи отговарят за различни микроуслуги. Няма ясно правило колко голяма може да бъде една микроуслуга. В предишния пример услугата на потребителския профил може да отговаря за данни като потребителското име и паролата на потребител, но също така и за домашния адрес, изображението на аватара, любимите и т.н. Може също да бъде опция за разделяне на всички тези отговорности на още по-малки микроуслуги. Архитектурата на микроуслугите е по-лесна за мащабиране, тъй като можете да се мащабират само определени микроуслуги.

5. Атрибути на качеството. Примери

При разработката на даден софтуерен продукт се гонят и т.н. атрибети на качеството или качествени параметри като сигурност, модулност, разбираемост, леснота на поддръжката, скалируемост на кода, висока производителност, бързодействие и други важни от гледна точка на бизнеса качества. Фактически и

самите шаблони допринасят за това. Това са на практика едни от най-важните критерии, по които се избира шаблон или съвкупност от шаблони. Шаблонният език, по който се води избора на правилна комбинация от шаблони, е в зависимост от избраните качествени параметри, както и от функционалните изисквания.

Софтуерните проекти стават сложни, по-големи, по-интегрирани и се реализират чрез използването на няколко разновидности на технологии. Тези различни технологии трябва да се управляват и организират, за да доставят качествен продукт. Атрибутите за качество обикновено се оценяват и анализират на ниво архитектура, а не на ниво код. Софтуерната архитектура придобива все по-голяма значимост, тъй като софтуерната индустрия има огромна нужда от стабилни системи, които поддържат голяма количество нужна функционалност, осигуряват високо качество и устойчивост на софтуерните продукти.

Измерването на различните аспекти на софтуерната архитектура, метриките и показателите й, оформят групата от атрибути за качество на всяка софтуерната архитектура. Създаването на добри архитектури обикновено идва на цената на значителната първоначална инвестиция. Ето защо осигуряването на добро качество през целия софтуерен проект е от първостепенно значение. Измерването е от решаващо значение и организациите се стремят да измислят значими мерки, които да показват напредък и резултати. Измерването в софтуерната архитектура е решаващ фактор за оценка на характеристиките на качеството на софтуера.

Възможността един софтуер да бъде лесно поддържан определя степента, до която софтуерът се разбира, рефакторира или подобрява. Трябва да е лесно да се имплементират нови промени, било за добавяне на нова функция или за отстраняване на грешки. Поддръжката е лекотата, с която може да бъде модифициран софтуера, да бъде лесно адаптиран за други цели или да може да бъде прехвърлите от един екип за разработка в друг. Съответствието на софтуерните архитектурни правила и последователността на приложението и разработката на една система, се комбинират, за да направят софтуера лесен за поддръжка.

Разрастването и развитието на един проект е неизбежна стъпка от жизнения цикъл на всеки софтуер. Възможността за разширяване е способността на софтуерната архитектура да се справи с добавянето на нови функционалности и компоненти. Това е много ценен аспект от атрибутите за качество най-вече при нуждата от бързо развитие, тъй като работата по един софтуер далеч не приключва след първата му релизната и доставена на клиента версия.

Простота на една архитектура може да бъде определена като добра структура. Важно е тя да бъде разбираема за всички, като целта е да взаимовръзките между различните компоненти да бъдат структурирани по най-добрия начин. Лесното разбиране на архитектурата от трети страни, неработещи по проекта, би било от изключително значение при вграждането на функционалности от външни библиотеки.

Производителността показва реакцията на системата по време на извършване на определени действия за определен период. Включва архитектурни метрики като латентност: време, прекарано в отговор на събитие, капацитет на работа: броят на събитията, които се случват в определен момент от времето. Обемът работа и натовареност, който може да поеме един софтуер има огромно значение за клиентите, които той би обслужвал. Неспособността му да се справи, би била огромен проблем. Ето защо преценката и ранната оценка на екипа работещ по системата са важни, за да може тя да работи както се очаква.

Мащабируемост е способността на системата да се справя с увеличаване на товара, без да намалява производителността, или възможността за бързо увеличаване на товара. Ключовите показатели за измерване на този атрибут са: системата да позволява хоризонтално мащабиране и времето, необходимо за увеличаване на мащабирането. Ограниченията за мащабиране също влизат в характеристиките на лесно мащабируемата система: броя на сървърите или капацитета на мрежата. С мащабируемостта се свързва и възможността за увеличаване на броя на транзакции или количеството съдържание в една система.

Риск от неизправност на софтуера и стабилност на програмата при излагане на неочаквани условия е в основата на надеждният софтуер. Той трябва да има минимален престой, добра цялост на данните и без грешки, които пряко засягат потребителите.

Един от най-ключовите фактори при разработката на един софтуерен продукт, особено днес, това е сигурността. Тя оценява колко добре дадено приложение защитава информацията срещу риска от софтуерни пробиви. Например количеството и тежестта на уязвимостите, открити в дадена софтуерна система, са показатели за нейното ниво на сигурност. Също така сигурността може да бъде измервана, като преценка за това колко време отнема да се поправят уязвимости на софтуера.

Оперативна съвместимост отговаря за работата и предаването на данни и обмена им с други външни системи. Добре проектираната система улеснява интеграцията със системи на трети страни. За подобряване на оперативната съвместимост, можете да се използват добре проектирани външни интерфейси, системи за стандартизация и др.

Жизнения цикъл на една софтуерната архитектура разкрива измеренията, по които се очаква да се развива система. Планът на разработка е важен, що се отнася до поставяне на срокове пред трети страни и клиенти. Той е свързан и с курса и скоростта на доставка на продукта, т.е. това колко често се изпращат нови версии на софтуера до клиентите. Тъй като новата версия на софтуера обикновено идва с подобрения, които пряко въздействат върху потребителите, можете да се заключи, че високото нива на доставка, съответстват на по-качествен софтуер за клиентите.

6. Влияние на софтуерната архитектура върху атрибутите на качеството

Софтуерната архитектура е основата, която помага за изграждането на система, която отговаря на бизнес изискванията. Организациите често мислят, че проектирането на софтуер се състои само разработването на неговите възможности и поведение, това обаче не е така. Потребителите са засегнати не само от липсата на подходяща функционалност, но и от липсващите необходими атрибути за качество на софтуера, които влияят върху жизнеспособността на всяко софтуерно решение. Когато системата не е надеждна, сигурна или мащабируема, тя неизбежно ще се провали по същия начин, както ако бъде забравено критично функционално изискване.

Качественият софтуер изисква висока степен на проверка. Тестването играе ключова роля при намирането на грешки. Високата възможност за тестване помага за ранното откриване на потенциални рискови ситуации. Добрата тестова инфраструктура прави тестването по-лесно, а системи по-малко вероятна да съдържат грешки, когато се изпращат до крайни потребители. Колкото по-трудно е да се провери функционалността на системата, толкова по-трудно е осигуряването на качество, а времето за внедряване в производството значително се увеличава. Основните показатели за този атрибут са процентът на покритие с интеграционни тестове, модулни тестове и др.

7. Документиране на софтуерна архитектура

Основен аспект за успеха на софтуера е способността да бъде описан функционално и да бъде достъпен за потребителите. Документацията служи за комуникация на архитектурата с други страни, които биха я използвали, подпомага екипа за разработка и обучението на членовете на екипа, улеснява анализа на архитектурата, позволява повторна употреба на вече приложени архитектурни знания. Тя обхваща всички писмени документи и материали, занимаващи се с разработването и използването на софтуерен продукт. Всички продукти за разработка на софтуер, независимо дали са създадени от малък екип или голяма корпорация, изискват някаква свързана документация. Различните видове документи се създават през целия жизнен цикъл на разработката на един софтуер. Съществува документация, която обяснява функционалността на продукта, унифицира свързаната с проекта информация и дава възможност за обсъждане на всички важни въпроси, възникващи между заинтересованите страни и разработчиците.

Документацията носи стойност за себе си и архитектурата, която описва. Тя служи като място за съхраняване на резултатите от дизайнерските решения при тяхното вземане. Една добре обмислена схема на документация може да накара

процеса на проектиране да върви много по-гладко и систематично. Документацията помага на архитекта, докато архитектурата е в ход.

Основната цел на ефективната документация е да гарантира, че разработчиците и заинтересованите страни се насочват в една и съща посока, за да постигнат целите на проекта. За постигането им съществуват две основни категории софтуерна документация:

- Документация за продукта
- Документация за процеса

Документацията на продукта описва продукта, който се разработва, и предоставя инструкции как да изпълнявате различни задачи с него. Документацията за продукта може да бъде разделена на системна документация и потребителска документация. Системната документация представлява документи, които описват самата система и нейните части. Той включва документи за изисквания, дизайнерски решения, описания на архитектурата, изходен код на програмата и ръководства за помощ. Потребителската документация обхваща ръководства, които са изготвени главно за крайни потребители на продукта и системни администратори. Потребителската документация включва уроци, ръководства за потребителя, ръководства за отстраняване на неизправности, инсталационни и справочни ръководства.

Документация за процеса, от своя страна, представлява всички документи, създадени по време на разработването и поддръжката, които описват самия процес. Честите примери за документация на процеса са планове на проекти, графици на тестове, доклади, стандарти, бележки за срещи или дори бизнес кореспонденция.

Основната разлика между документацията за процеса и продукта е, че първата записва процеса на разработване, а втората описва продукта, който се разработва. Съответно към втората категория се вписва и документацията за проектиране на софтуерна архитектура. Тя включват основните архитектурни решения като принципи на архитектурата и дизайна й, описание на потребителските изисквания, подробности за решението, диаграмно представяне и др.

8. Оценка на софтуерна архитектура

Важна стъпка при проектирането на висококачествена софтуерна архитектура е тя да премине през процес на преглед. Прегледи на архитектурата също могат да се провеждат, когато организацията придобие софтуер или за сравняване на архитектури. Прегледът ще определи дали функционалността, изискванията и сценариите за качество на атрибутите могат да бъдат удовлетворени от софтуерната архитектура. Оценката на архитектурата помага на екипа да намери грешки и да ги

поправи възможно най-рано. Това може значително да намали количеството усилия, необходими за отстраняване на дефект, и може да помогне за избягване на допълнителни преработки.

Изборът на архитектурен модел е силно обвързан с въпроси като това как може да сме сигурни, че избраната софтуерна архитектура за даден софтуер, е правилната и подходяща, и няма да доведе до проблеми, а вместо това ще проправи пътя през гладкото развитие и успешния продукт.

Основата на всяка софтуерна система е нейната архитектура. Архитектурата ще позволи или изключи почти всички атрибути за качество на системата. Модифицируемост, производителност, сигурност, наличност, надеждност - всичко това е предварително сглобено, след като бъде определена архитектурата. Никаква настройка или хитри трикове за внедряване няма да изтръгнат някое от тези качества от лошо архитектурна система. Архитектурата е залог за успеха или провал на една системата. Това води до необходимост от начин за ранно диагностициране и оценяване на работоспособността и издръжливостта ѝ. Късното откриване на проблеми със софтуера би коствало много разходи както фирмата клиент, така и на фирмата разработчик.

Оценката на софтуерната архитектура е изградена около набор от четири метода, всички разработени в Института по софтуерно инженерство, които могат да бъдат приложени към всяка софтуерно интензивна система:

- SAAM: Метод за анализ на софтуерната архитектура
- ATAM: Методът за архитектурен анализ на компромисите
- ADR: Метод за активен преглед на дизайна
- ARID: Методът за активни рецензии за междинни дизайни

Методите като група имат солидно родословие, като се прилагат от години в десетки проекти с всякакви размери и в голямо разнообразие от домейни. С тези методи е дошло времето да включим оценката на софтуерната архитектура като стандартна стъпка от всяка парадигма на разработката. Оценките представляват разумно усилие за намаляване на риска и са относително евтини. Те плащат за себе си по отношение на скъпи грешки и избягвани безсънни нощи. Архитектурата дефинира компонентите (като модули, обекти, процеси, подсистеми, компилационни единици и т.н.) и съответните връзки (като повиквания, изпращане на данни към, синхронизиране с, използване, зависи от, екземпляри, и много други) сред тях. Архитектурата е резултат от ранни дизайнерски решения, които са необходими преди група хора да могат съвместно да изградят софтуерна система. Колкото по-голяма или по-разпределена е групата, толкова по-жизненоважна е архитектурата.

Методът за анализ на софтуерната архитектура (SAAM) е един от първите документирани методи за оценка на софтуерни архитектури. Първоначалната цел на SAAM е била да оцени модифицируемостта на софтуерна система. По-късно бива

доразработена тази идея, за да оценява и различните атрибути на качество на софтуерната архитектура, включително надеждност, възможност за интегриране, възможност за разрастване и производителност. SAAM е метод, базиран на сценарий, и може да бъде ефективен начин за преглед на софтуерна архитектура. Сценарият е описание на взаимодействието между някакъв източник, като заинтересована страна, и софтуерна система. Той представлява някаква употреба или очаквано качество на софтуерна система и може да се състои от поредица от стъпки, подробно описващи използването или модификацията й.

Методът за архитектурен анализ на компромисите (ATAM) е друг метод за преглед на архитектурата, базиран на сценарии. ATAM е наследник на SAAM и го подобрява, като се фокусира върху прегледа на дизайнерските решения и атрибутите за качество. Основните роли на участниците по време на оценката на ATAM са екипът за оценка, лицата, вземащи решения по проекта, и заинтересованите страни. Екипът за оценка, който в идеалния случай трябва да бъде група, която е външна за софтуерния проект, се състои от ръководител на екип, ръководител на оценка и др. Според научният труд „ATAM: Method for Architecture Evaluation" съвместно със софтуерния инженерен институт „Carnegie Mellon", нейните автори, Марк Клейн, Пауло Клементс и Рик Казманцернс, предлагат следната характеристична дефиниция на архитектуреният метод за анализ на компромисите (ATAM):

„ ATAM е метод за анализ, организиран около идеята, че архитектурните стилове са основните определящи фактори за качество на архитектурата. Методът се фокусира върху идентифицирането на бизнес цели, които водят до цели за качество на качествата. Въз основа на целите на атрибутите за качество, ние използваме ATAM, за да анализираме как архитектурните стилове помагат за постигането на тези цели." [4]

По време на процеса на анализ свързан с ATAM се обсъждат възможни ограничения, системната функционалност и желаните нефункционални свойства. След това от тези заключения се създават атрибути за качество и бизнес сценарии. Накрая, заедно с архитектурни подходи и архитектурните проекти, тези сценарии се използват за създаване на анализ на компромисите, точки на чувствителност и рискове.

Метод за активен преглед на дизайна (ADR) е най-подходящ за архитектурни проекти, които са в процес на изпълнение. Този тип преглед на архитектурата е по-фокусиран върху прегледа на отделни раздели на архитектурата в даден момент, а не върху извършването на общ преглед. Процесът включва идентифициране на проблеми с дизайна и други грешки в архитектурата, така че те да могат да бъдат коригирани възможно най-бързо и рано в цялостния процес на проектиране. Една от основните характеристики на ADR е, че променя фокуса от по-общ преглед на

[4]ATAM: Method for Architecture Evaluation / Mark Klein, Paul Clements, Rick Kazmancerns. - August 2000, 2020 (Software Engineering Institute) - p.7

цялата архитектура, на поредица от по-фокусирани прегледи. Създава се комуникация между рецензентите и архитектите на системата.

Методът за активни рецензии за междинни дизайни (ARID) е метод за преглед на архитектурата, който съчетава ADR с ATAM. Този хибриден метод взема от ADR подхода с фокуса върху прегледа на софтуерната архитектура, докато тя е в ход, и акцента върху активната участие на рецензенти. Той съчетава това с ATAM подхода за фокусиране върху сценариите за атрибути на качеството. Целта е да се осигури ценна обратна връзка за жизнеспособността на софтуерната архитектура и да се открият всички грешки и недостатъци с нея.

Прегледът на софтуерна архитектура е важен, за да се определи дали архитектурата ще отговаря на нуждите на системата. Това е стъпка в софтуерната разработка, която би спестила много разходи и проблеми със софтуера, за това е важно да бъде избран най-правилният подход.

9. Заключение

Софтуерната архитектура е вид дизайн, който се занимава с мащабни решения и макроскопични елементи (например модули, компоненти и връзки) в една софтуерна система. Тя е важна поради голямо разнообразие от технически и нетехнически причини, като основната е, че тя е скелета на всяка система, влияе върху нейните качества и функционалност. Добрите решенията, взети в дадена архитектура, позволяват по-лесно надграждане и управление на промените, с развитието на една система. Анализът на архитектурата позволява ранно прогнозиране на качествата на системата, а добре документираната архитектура подобрява комуникацията между заинтересованите страни.

Архитектурата е носител на най-ранните и следователно най-фундаментални, най-трудни за промяна дизайнерски решения. Тя определя набор от ограничения при последващото внедряване и диктува структурата на организацията. По време на разработка на софтуерните идеи има възможност разработчиците да намаля сложността на дизайна на системата и да предотвратят бъдещи проблеми, избирайки подходящ модел на изграждането й.

10. Използвана литература

1. Software Architecture In Practice / Len Bass, Paul Clements, Rick Kazmancerns. - 3rd ed., 2013 (Software Engineering Institute) - p.4
2. Fundamentals of Software Architecture / Mark Richards, Neal Ford. - 1st ed., 2020 - p.7

3. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnsonq John Vlissudes . - November, 1994 – p.13
4. ATAM: Method for Architecture Evaluation / Mark Klein, Paul Clements, Rick Kazmancerns. - August 2000, 2020 (Software Engineering Institute) - p.7

5. Beyond Software Architecture Creating and Sustaining Winning Solutions / Luke Hohmann .- 1st ed.
6. Practical Software Architecture Moving from System Context to Deployment / Tilak Mitra .-2016
7. Software Architecture Patterns / Mark Richards .- 2015
8. Software Architecture: a Roadmap / David Garlan .- School of Computer Science (Carnegie Mellon University)
9. Documenting Software Architectures Views and Beyond / David Garlan, Felix Bachmann, James Ivers, Reed Little, Judith Stafford, Len Bass, Paul Clements, Paulo Merson, Robert Nord .- 2nd ed.
10. Software Architecture: An Executive Overview / Paul C. Clements, Linda M. Northrop .- February, 1996 (Technical Report)