Развој софтвера 2

3.4. Предуслови: Архитектура

Софтверска архитектура је део софтверског дизајна на високом нивоу, тј. оквир који држи детаљније делове дизајна на окупу. Архитектура је још позната као “системска архитектура”, “дизајн високог нивоа” или “дизајн највишег нивоа”. Архитектура се обично описује у једном документу који назива “спецификација архитектуре” или “дизајн највишег нивоа”.

Неки аутори праве разлику између архитектуре и дизајна високог нивоа. По њима, архитектура се односи на ограничења дизајна која се примењују на нивоу целог система, док дизајн високог нивоа означава ограничења дизајна која се примењују на нивоу подсистема или групе класа, а не на нивоу целокупног система.

С обзиром да се презентација односи на конструкцију, овде се не описује како се развија архитектура система, већ се фокусира на то како се одређује квалитет развијене архитектуре. Будући да је архитектура за корак ближа конструкцији у односу назахтеве, то ће разматрање архитектуре бити детаљније од разматрања захтева.

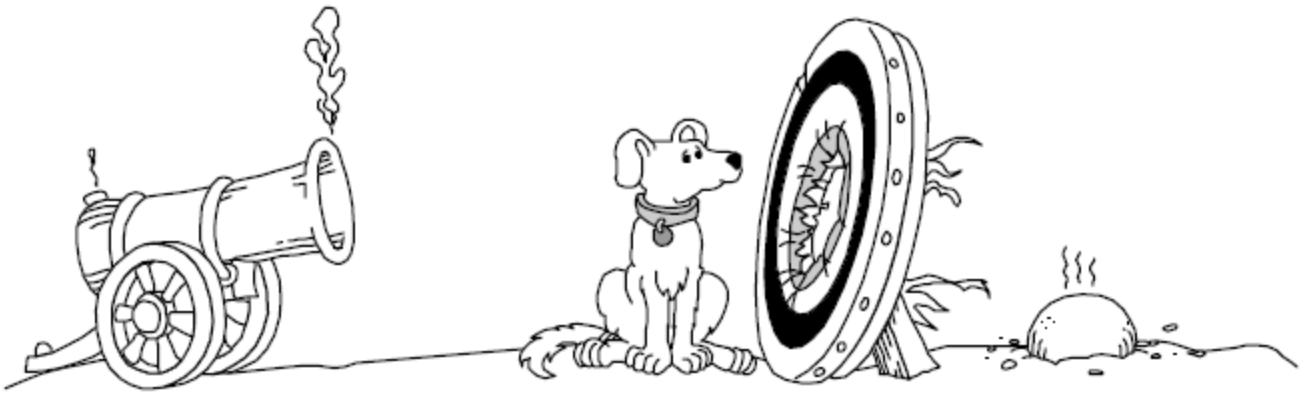
3.4. Предуслови: Архитектура

Зашто архитектура представља предуслов? Зато што квалитет архитектуре одређује концептуални интегритет система, који заузврат одређује крајњи квалитет система. Добро промишљена архитектура обезбеђује структуру која је потребна за управљање концептуалним интегритетом целог система, почев од највиших нивоа, па надоле.

Архитектура обезбеђује вођење програмерима на нивоу детаљности који одговара вештини програмера и самом послу. Она партиционише посао тако да већи број програмера из више развојних тимова може да ради независно.

Добра архитектура обезбеђује лаку конструкцију. Лоша архитектура чини да конструкцију скоро немогућом.

3.4. Предуслови: Архитектура



Слика 3-5
Када нема добре архитектуре софтвера, тада имамо прави програм али лоше решење, па може бити немогућа успешна конструкција.

Јако је скупо да се промене у архитектури врше током конструкције или касније. Време потребно за поправку грешке настале у архитектури је истог реда као време потребно за поправку грешке настале код захтева, што је много више од времена потребног за исправак грешке настале при кодирању. Исто као код захтева, чак и мале промене архитектуре имају велике последице. Без обзира да ли су промене у архитектури настале како би се исправила грешка или из потребе унапређења софтвера, много је боље да се потребне промене идентификују што је пре могуће.

3.5. Типичне компоненте архитектуре

Добре системске архитектуре имају много заједничких компоненти. Ако сами градите цео систем, тада ће се ваш рад на архитектури преклапати са радом на детаљном дизајну. У таквом случају, треба бар промислити о свакој архитектонској компоненти. Ако се ради на систему чију је архитектуру осмислио неко други, требало би да можете да будете у стању да пронађете важне компоненте система без њушења трагова, коришћења лупе и друге опреме Шерлока Холмса. Шта год било у питању, следе архитектонске компоненте које треба размотрити.

3.5.1. Организација програма

Системска архитектура прво захтева преглед са најширим описом система. Ако нема таквог прегледа, онда се суочавате са тешкоћом грађења кохерентне слике из хиљаду детаља који потичу од неколико десетина (до неколико стотина) појединачних класа. Ако је систем мала слагалица од 12 делова, онда њу може да сложи и трогодишње дете. Већ је теже склопити слагалицу од 12 класа или 12 подсистема, а ако нисте у могућности да склопите такву слагалицу тада нећете разумети на који начин класе које развијате доприносе раду система.

3.5. Типичне компоненте архитектуре

У документу који описује архитектуру се морају налазити и докази да су размотрене алтернативе изабраној организацији система, као и разлози због којих је баш таква организација добила предност у односу на разматране алтернативе. Јако је фрустрирајуће радити на класи када изгледа да није добро размотрена улога те класе у систему. Описивањем организационих алтернатива, документ о архитектури даје разлоге за изабрану организацију система и показује да је свака од класа пажљиво размотрена. Разматрање пракса дизајнирања указује да су разлози за изабрани дизајн бар онолико важни колико и само праћење изабраног дизајна.

Документ о архитектури треба да дефинише главне градивне блокове програма. Сваки од градивних блокова може бити било једна класа, било подсистем који садржи већи број класа - у зависности од величине програма који се развија. Градивни блок је или класа, или колекција класа тј. рутина које заједно раде и обезбеђују функционалност високог нивоа, као што је интеракција са корисником, приказ веб страна, интерпретирање наредби, енкапсулација пословних правила, приступ подацима, итд.

Ако нешто не можете да објаните шестогодишњем детету, онда ни ви сами то не разумете у потпуности. — Алберт Ајнштајн

3.5. Типичне компоненте архитектуре

Свака од карактеристика која је побројана у захтевима треба да буде покривена са бар једним градивним блоком. Ако је функционалност захтевана од два или више градивних блокова, тада захтеви треба да буду кооперативни, а никако не смеју да буду у конфликту.

Треба да буде јасно дефинсано за шта је одговоран сваки од градивних блокова. Градивни блок треба да има једнну област одговорности и треба да зна што је могуће мање о областима одговорности других градивних блокова. Минимизацијом информација о томе шта један градивни блок зна о другим градивним блоковима, постиже се локализација информација о дизајну у унутрашњост једног градивног блока.

Треба адекватно дефинисати правила за комуникацију за сваки од градивних блокова. Архитектура би требало за дати градивни да опише које од осталих градивних блокова тај блок може директно да користи, које може индиректно да користи, а које уопште не сме да користи.

3.5. Типичне компоненте архитектуре

3.5.2. Најважније класе

Документ о архитектури треба да одреди најважније класе које ће се користити. Он треба да идентификује одговорности најважнијих класа и на који ће начин те класе да имају интеракцију са другим класама. Документат о архитектури треба да садржи опис хијерархије класа, преласке између стања и перзистенцију објеката. Ако је систем довољно велик, архитектура треба да опише како су класе организоване у подсистеме.

Документ о архитектури треба да опише и другачије дизајне класа који су били размотрени, као и разлоге због којих је предност дата изабраном дизајну у односу на друге разматране алтернативе.

Документ о архитектури не треба да специфицира сваку класу у систему, већ се обично користи “80/20 правило”: специфицира се 20% класа преко којих се реализује 80% понашања система.

3.5. Типичне компоненте архитектуре

3.5.3. Дизајн података

Документат о архитектури треба да опише дизајне најважнијих табела и датотека. Он треба да опише и разматране алтернативе и да оправда донесене одлуке тј. направљене изборе. На пример, ако апликација одржава списак идентификатора клијената и ако су архитекти одлучили да се ти идентификатори чувају у једноструко повезаној листи са секвенцијалним приступом, документ треба да објасни зашто је изабрани начин чувања података бољи од листе са директним приступом, стека или хеш-табеле. Такве информације обезбеђују слику о размишљању архитеката која је јако корисна током конструкције. Иста та слика је од непроцењиве помоћи током одржавања, јер без се ње налазите у ситуацији као да гледате филм на непознатом језику а да нема титлова.

Обично се подацима директно приступа само из једног подсистема или класе, осим код класа и рутина које допуштају приступ подацима на контролисан и апстрактан начин.

Архитектура треба да специфицира организацију на високом нивоу и садржај свих база података. Она треба да објасни зашто је једна база боља од више (или обратно), да идентификује могуће интеракције са другим програмима који приступају подацима, објасни који ће погледи бити креирани над подацима итд.

3.5. Типичне компоненте архитектуре

3.5.4. Пословна правила

Ако архитектура зависи од конкретних пословних правила (енг.bBusiness rules), она треба да их идентификује и да опише утицај који та правила имају на дизајн целокупног система.

На пример, претпоставимо да се од система захтева да поштује пословно правило по коме информације о кориснику не смеју бити неажурне дуже од 30 секунди. У том случају, мора да се прецизно објасни утицај који на архитектонски приступ има захтев да информације буду ажурне и синхронизоване.

3.5. Типичне компоненте архитектуре

3.5.5. Дизајн корисничког интерфејса

Понекад се кориснички интерфејс специфицира у време креирања захтева. Ако то није био случај, онда се кориснички интерфејс треба специфицирати при креирању софтверске архитектуре. Документ о архитектури треба да специфицира најважније елементе који се односе на формат веб стране, графички кориснички интерфејс, интерфејс командне линије итд. Пажљиво дефинисање корисничког интерфејса прави разлику између програма који се свиђа корисницима и програма који нико не користи.

Архитектура треба да буде модуларна, тако да нови кориснички интерфејс може да буде замењен а да та замена не утиче на пословну логику нити на делове програма који исписују излазне податке. На пример, архитектура треба да омогући релативно лако искључење групе класа са интерактивним интерфејсом и укључивање групе класа које “раде” преко командне линије. Таква могућност је често корисна, зато што су интерфејси командних линија погодни за тестирање софтвера на нивоу јединица (енг. unit test) и на нивоу подсистема (енг. subsystem test).

3.5. Типичне компоненте архитектуре

3.5.6. Улаз/Излаз

Улаз/Излаз представља још једно подручје коме треба посветити пажњу приликом креирања архитектуре. Архитектура треба да специфира да ли се, ако се уопште читају подаци из датотеке користи схема читања “гледај напред” (енг. look-ahead), “гледај позади” (енг. look-behind), или “тачно на време” (енг. just-in-time).

Креирани документ о архитектури треба и да опише на ком нивоу ће се детектовати грешке Улаза/Излаза: на нивоу поља, на нивоу слога, на нивоу тока података или на нивоу целе датотеке.

3.5. Типичне компоненте архитектуре

3.5.7. Управљање ресурсима

Документ о архитектури треба да опише план за управљање оскудним ресурсима као што су конекције према бази података, нити и руковаоци датотеком. У областима апликација где постоји меморијско ограничење, као што су развој драјвера за уређаје и развој уграђених система, је приликом дефинисања архитектуре неопходно разматрати управљање меморијом. Документ о архитектури треба да процени ресурсе који се користе у нормалним случајевима, као и ресурсе који се користе у екстремним случајевима.

У једноставном случају, процене треба да покажу да су потребни ресурси сасвим унутар капацитета имплементационог окружења које намеравамо да користимо.
У сложенијем случају, може се захтевати да апликација сама активније управља са сопственим ресурсима. Ако је то случај, тада менаџер ресурса треба да буде дефинисан током креирања архитектуре, при чему се дефинисању менаџера ресурса треба посветити иста пажња као другим деловима система.

3.5. Типичне компоненте архитектуре

3.5.8. Сигурност

Документ о архитектури треба да опише приступ према сигурности на нивоу дизајна и на нивоу кода. Ако модел претњи није био до тада изграђен, треба га направити током осмишљавања архитектуре.

Правила за запис кода (енг. coding guidelines) треба да буду развијена тако да се имају у виду сигурносне импликације, што обухвата приступе за руковање баферима, приступе за рад са подацима за које нема поверења (као што су улазни подаци корисника, колачићи, конфигурациони подаци, други спољашњи интерфејси итд.), енкрипцију, ниво детаља које садрже поруке о грешци, заштиту тајних података који се налазе у меморији, итд.

3.5. Типичне компоненте архитектуре

3.5.9. Перформансе

Ако постоји забринутост око перформанси, тада циљне перформансе треба да буду специфициране током креирања документа о захтевима. Циљне перформансе се могу односити и на брзину и на коришћење меморије.

Документ о архитектури треба да обезбеди процене и да објасни зашто архитекти верују да су циљне перформансе достижне.
Ако у неким областима постоји опасност да се постављени циљ не испуни, документ о архитектури треба то јасно да искаже.
Ако неке области захтевају да се користе специфицирани алгоритми или типови података како би биле достигнуте циљне перформансе, документ о архитектури треба то јасно да искаже.

Документ о архитектури може такође да укључи одређивање просторног и временског буџета за сваку од класа или објеката.

3.5. Типичне компоненте архитектуре

3.5.10. Скалабилност

Скалабилност је способност система да расте, како би могао да изађе у сусрет будућим захтевима. Документ о архитектури треба да опише како ће се систем односити према порасту броја корисника, порасту броја сервера, броја мрежних чворова, величине базе података, величине обима трансакција итд.

Ако се не очекује да ће систем расти тј. ако се не разматра скалабилност, онда у документу о архитектури тај став треба експлицитно написати.

3.5. Типичне компоненте архитектуре

3.5.11. Интероперабилност

Ако се очекује од система да дели податке или ресурсе са другим софтвером или хардвером, тада архитектура треба да опише како ће то бити постигнуто.

3.5. Типичне компоненте архитектуре

3.5.12. Интернационализација/Локализација

Интернационализација (енг. Internationalization, још позната као I18N) је техничка активност припреме програма да подржава већи број језика.
Интернационализација се преплиће са локализацијом. Локализација (енг. Localization, тј. L10n) је активност превођења програма тако да подржава конкретан локални језик.
Питања интернационализације привлаче пажњу у дефинисању архитектуре интерактивног система. Највећи број интерактивних система садржи на десетине и стотине питања, приказа статуса, порука које садрже помоћ, порука о грешкама итд.
Треба проценити ресурсе са знаковним низовима. Документ о архитектури треба да покаже како су разматрана типична питања која се односе на текст, знакове и кодне стране, на који ће се начин одржавати и мењати текст и како ће се преводити текст на друге језике а да утицај на код и на кориснички интерфејс буде најмањи могући. Документ о архитектури може да одлучи да се користи текст унутар кода, да се чува у класама и реферише на њега кроз интерфејс класе или да се смести у ресурсне датотеке. У том документу се треба описати која је опција изабрана, као и разлози за тај избор.

3.5. Типичне компоненте архитектуре

3.5.13. Обрада грешака

Испоставља се да је обрада грешака један од тежих проблема у модерном рачунарству, па се можемо себи приуштити да се са њим бавимо несистематски. Неки аутори су проценили да се чак до 90% програмског кода пише ради обраде изузтетака, поспремања при обради грешака, одакле следи да се само 10% кода односи на “нормалне” ситуације.

Чим се толико много кода бави руковањем са грешкама, јасно је да архитектура треба да специфицира стратегију за њихову обраду на конзистентан начин.

Руковање грешкама се често посматра на нивоу конвенција кодирања (не архитектуре), а има и случајева у којима се уопште не разматра. Ипак, с обзиром да његове импликације утичу на цео систем, најбоље га је посматрати на нивоу архитектуре.

3.5. Типичне компоненте архитектуре

Следе нека од питања која треба размотрити при дефинисању овог аспекта архитектуре:

● Да ли је обрада грешака корективна или се ограничава на откривање? Ако је корективна, тада програм може да покуша да се опорави од неких грешака. Ако је само ограничена на откривање, тада програм може наставити рад као да се ништа није догодило, или може да се заустави. У сваком случају, корисник треба да буде обавештен да је детектована грешка.
● Да ли је деткација грешака активна или пасивна? Систем може да активно предвиђа грешке (нпр. проверавајући валидност корисниковог уноса) или може пасивно да одговара на грешке онда када не може да их избегне (нпр. када комбинација уноса корисника доведе до нумеричког прекорачења). Другим речима, она може или да расчишћава пут или да чисти sa собом. У оба случаја избор алтренативе има импликације на кориснички интерфејс.
● Како програм прослеђује информацију о грешци? Када се открије грешка, може се одлучити да се одмах пониште подаци који су довели до грешке, одлучити да се грешка тестира као грешка уласком у стање обраде грешке, или да се чека завршетак свих обрада и да се тада информише корисник о томе да је током рада откривена грешка.

3.5. Типичне компоненте архитектуре

● Које су конвенције за руковање грешкама? Ако архитектура не специфицира јединствену конзистентну стратегију, тада се кориснички интерфејс појављује као конфузан колаж различитих интерфејса и различитих делова програма. Да би се избегло овакво понашање, архитектура треба да утврди конвенције за поруке о грешкама.

● На ком нивоу се рукује грешкама унутар програма? Грешкама се може руковати на месту уткривања, могу се прослеђивати класи sa руковање грешкама или се могу прослеђивати навише кроз ланац позива.

● Који је ниво одговорности у валидацији улазних података за сваку од класа? Да ли је свака од класа одфговорна за валидацију својих података, или постоји група класа одговорна за валидацију података у систему? Да ли класе на неком од нивоа могу претпоставити да су подаци које они прихватају чисти?

● Да ли се користи механизам за руковање изузецима који је уграђен у окружење, или се прави сопствени? Чак иако окружење подржава конкретан приступ за руковање грешкама, то не мора значити да је у нашем случају тај приступ најбољи.

3.5. Типичне компоненте архитектуре

3.5.14. Отпорност на отказе

Документа о архитектури такође треба да укаже и на очекивани ниво толеранције на отказе (енг. fault tolerance). Толеранција на отказе је скуп техника који увећава поузданост система тако што открива грешке, опоравља систем од грешака (уколико је то могуће) и зауставља лоше ефекте грешака (уколико је опоравак немогућ).

На пример, систем може на неки од следећих начина обезбедити да израчунавање квадратног корена броја буде отпорно на отказе:

1) Систем може чувати међурезултате и покушати поново када детектује отказ. Ако је први одговор погрешан, он се може вратити до тачке на којој је сигурно све било у реду, па наставити од те тачке.

2) Систем може да има помоћни код који се користи у случају када се дектектује отказ. У конкретном примеру, ако је први одговор погрешан, систем се може пребацити тако да користи, уколико је то потребно, алтернативну рутину за рачунање квадратног корена.

3.5. Типичне компоненте архитектуре

3) Систем може да користи алгоритам гласања. Може да има три класе са методима које рачунају квадратни корен, при чему сваки од метода то ради на другачији начин. По завршетку израчунавања на сва три начина, систем упоређује добијене резултате. У зависности од врсте толеранције која је уграђена у систем, он као крајњи резултат може дати аритметичку средину, медијану или нешто треће.

4) Систем може заменити погрешну вредност са неком лажном вредношћу за коју се зна да ће имати бенигни ефекат на остатак система.

Остали приступи обезбеђења толерантности на отказе обухватају промену система тако да у случају детекције грешке прелази у стање делимичне оперативности или у стање умањене функционалности. На тај начин он може да се сам исључи или да се аутоматски рестартује. Ови примери су, због сврхе илустровања, доста поједностављени. Отпорност на отказе је фасцинантна и сложена тема, која излази из опсега овог курса.

3.5. Типичне компоненте архитектуре

3.5.15. Изводљивост архитектуре

Дизајнери могу бринути о могућностима система који се гради да постигне циљне перформансе, да ради са ограничењима у ресурсима која су му наметнута, или да буде адекватно подржан од стране окружења за имплементацију.

Документат о архитектури треба да демонстрира да је систем технолошки изводљив. Ако неизводљивост у ма којој до области може довести до немогућности рада на пројекту, документат о архитектури треба да укаже како су ови елементи истраживани – кроз прототипове који служе за проверу концепта, кроз истраживање или на други начин. Ови ризици треба да се разреше пре него што конструкција крене пуном снагом.

3.5. Типичне компоненте архитектуре

3.5.16. Превише интензиван развој (енг. overengineering)

Робусност је особина система да настави са радом чак и када се догоди грешка. Често докуменат о архитектури специфицира и робуснији систем наго што се тражи у захтевима. Један од разлога за то је што систем састављен од делова који су минимално робусни може бити мање робустан у целости него што је захтевано. У софтверској индустрији, ланац није онолико слаб колико је слаба његова најслабија карика – он је слаб онолико колико износи производ слабости свих слабих карика.

Архитектура треба јасно да укаже да ли при развоју програмери треба да теже повећаној интензивности или да се орјентишу на прављење најједноставније ствари коа задовољава услове и која функционише.

Одређивање приступа у овом аспекту је веома важно, зато што многи програмери, због професионалног поноса, аутоматски превише интензивно развију своје класе. Експлицитним постављањем очекивања у деокумент о архитектури, може се избећи појава да једна група класа буде изузетно робусна, а друге да буду скоро па неадекватно робусне.

3.5. Типичне компоненте архитектуре

3.5.17. Дилема између куповине и градње

Најрадикалније решење у изградњи софтвера је да се софтвер уопште не гради, већ да се уместо тога купи. Могуће је купити контроле графичког интерфејса, СУБП, компоненте за графике и дијаграме, компоненте за сигурност и енкрипцију, алате за раширене табеле, Интернет комуникационе компоненте, итд.

Једна од највећих предности програмирања у модерном окружењу је количина функционалности која је аутоматски на располагању: графичке класе, менаџери дијалога, руковаоци мишем и тастатуром, код који аутоматски ради са било којим монитором и штампачем, итд.

Ако архитектура не користи купљене компоненте тј. Компоненте на располагању, онда треба описати начин на који самостално изграђене компоненте треба да надмаше компонете које су већ на располагању.

3.5. Типичне компоненте архитектуре

3.5.18. Одлуке о поновном коришћењу (енг. reuse)

Ако план развоја захтева коришћење софтвера који већ постоји, тада документ о архитектури треба да објасни како софтвер који се поново користи треба да буде у сагласности са осталим археолошким циљевима (уколико се та сагласност и захтева).

3.5. Типичне компоненте архитектуре

3.5.19. Стратегија за промене

Будући да је изградња софтверског производа истовремено и процес учења и за програмере и за кориснике, може се очекивати да ће током свог развоја производ да се мења.

Промене проистичу из променљивих типова података и формата датотека, из промењене функционалности, из нових карактеристика итд. Промене могу бити нове способности које могу настати из планираних проширења, или могу бити могућности које нису укључене у прву верзију система. Према томе, један од највећих изазова са којим се суочава архитекта софтвера је креирање архитектуре која је довољно флексибилна да се може прилагодити очекиваним променама.

Архитектура треба јасно да опише статегију за руковање променама. Документ о архитектури треба да прикаже да су размотрена могућа проширења и да су она проширења која су највероватнија истовремено и најлакша за имплементацију. Ако се очекују промене у улазним или излазним форматима, у стилу интеракције са корисником, или у захтевима процесирања, архитектура треба да покаже да су све те промене биле предвиђене и да ефекти ма које од њих бивају ограничени на мали број класа.

3.5. Типичне компоненте архитектуре

Архитектов план за промене може бити једноставан, као што је постављање броја верзије у датотеке са подацима, резервисање поља за будуће коришћење, или дизајн датотека тако да се лако могу додати нове табеле. Ако се користи генератор кода, тада архитектура треба да покаже да су предвиђене промене у оквиру способности генератора кода.

Архитектура треба да укаже на стратегије које се користе ради пролонгирања обавезивања. На пример, архитектура мора да специфицира да ли ће се користит техника управљања табелом или if тестови тј. “питалице” који су унесени у код. Та архитектура може специфицирати да се подаци за табелу чувају у спољашњој датотеци пре него да буду кодирани унутар програма, што допушта да се програм мења без његове поновне компилације.

Грешке у дизајну су често веома суптилне и оне еволуирају када су заборављене ране претпоставке а нове карактеристике Du нови начини коришћења се додају у систем.

3.5. Типичне компоненте архитектуре

3.5.20. Општи квалитет архитектуре

Добра спецификација архитектуре се ларактерише разматрањима о класама у систему, о информацијама које су сакривене од других класа и о разлозима за укључивање и искључивање свих могућих алтернатива у дизајну.

Архитектура треба да буде углачана концептуална целина са неколико додатних ад-хок дефиниција. Централна теза најпопуларније књиге у софтверском инжењерству, *Mythical Man-Month*, је да је есецијални проблем са великим системима у одржавању њиховог концептуалног интегритета.

Добра архитектура треба да се уклопи у проблем. Када посматрате архитектуру, треба будете задовољни како решење лако и природно изгледа. Не сме да делује да је архитектура решења провизорно (“помоћу штапа и канапа”) провезана уз проблем.

Можда вам је већ познато на које се начине архитектура мења током развоја система. Свака од промена треба да се чисто и прецизно уклопи у основни концепт. Дизајн система чији је примарни циљ могућност лаке модификације се разликује од дизајна система у коме је циљ бескомпромисно постизање преформанси - чак и у случају да оба система имају исту функцију.

3.5. Типичне компоненте архитектуре

Архитектура треба да опише мотивацију за сваку од најважнијих одлука. Чувајте се оправдања типа “ми смо одувек радили на овај начин”.

Анегдота говори како је жена хтела да спреми говеђи рибић по рецепту који се са колена на колено преносио у мужевој породици. Њен супруг је описао начин припремања: узме се рибић, посоли се и побибери, одсеку му се оба краја, покрије се и стави се у већ загрејану рерну.
Жена је питала: Зашто се одсецају крајеви рибићу?
Супруг је одговорио: Не знам. Увек смо тако радили. Чекај да питам моју маму.
Супруг је позвао своју маму, питао је и она је одговорила: Не знам. Увек смо тако радили. Чекај да питам твоју баку.
Супругова мама је позвала баку, која је одговорила: Не знам зашто ви тако радите. Ја радим тако зато што је рибић превише велики да стане у посуду.

Добра архитектура софтвера је у великој мери независна од рачунара и независна од језика.

Наравно, не може се потпуно игнорисати окружење конструкције. Тиме што смо у највећој могућој мери независни од окружења, лакше ћемо се одупрети искушењу да превише интензивно направимо архитектуру система или да радимо посао који се може боље урадити у фази конструкције (пример је креирање оперативног система Windows NT). Ако се програм извршава само на једној врсти рачунара или на једном језику, тада горње упутство не важи.

3.5. Типичне компоненте архитектуре

Архитектура треба да постави линију између недовољног специфицирања система и превише интензивног специфицирања система. Ниједан део архитектуре не треба да добије више пажње него што заслужује, иначе ће да буде превише интензивно дизајниран (енг. over-designed). Дизајнери не треба да посвећују пажњу једном делу система на уштрб неког другог дела.

Архитектура треба да прикаже све захтеве без “послуживања са златног овала” тј. не треба да буду обухваћени елементи који нису захтевани.

Архитектура треба да експлицитно идентификује ризична подручја. Она треба да објасни зашто су та подручја ризична и који су кораци предузети у циљу минимизације ризика.

На крају, не би смели да будемо нејасни у опису ма ког дела архитектуре. Документ о архитектури не би треба да садржи текст који је стављен у документ да би шеф био задовољан. Документ не би требао да садржи делове који су нам тешко разумљиви. Ми треба да имплеметирамо систем, па ако нама тај документ нема смисла, онда се поставља питање како ћемо имплементирати систем на основу тог документа?

3.6. Листа за проверу за архитектуру

3.6.1. Specific Architectural Topics

• Is the overall organization of the program clear, including a good architectural overview and justification?
• Are major building blocks well defined, including their areas of responsibility and their interfaces to other building blocks?
• Are all the functions listed in the requirements covered sensibly, by neither too many nor too few building blocks?
• Are the most critical classes described and justified?
• Is the data design described and justified?
• Is the database organization and content specified?
• Are all key business rules identified and their impact on the system described?
• Is a strategy for the user interface design described?
• Is the user interface modularized so that changes in it won’t affect the rest of the program?

3.6. Листа за проверу за архитектуру

- Is a strategy for handling I/O described and justified?
- Are resource-use estimates and a strategy for resource management described and justified?
- Are the architecture's security requirements described?
- Does the architecture set space and speed budgets for each class, subsystem, or functionality area?
- Does the architecture describe how scalability will be achieved?
- Does the architecture address interoperability?
- Is a strategy for internationalization/localization described?
- Is a coherent error-handling strategy provided?
- Is the approach to fault tolerance defined (if any is needed)?
- Has technical feasibility of all parts of the system been established?
- Is an approach to overengineering specified?
- Are necessary buy-vs.-build decisions included?
- Does the architecture describe how reused code will be made to conform to other architectural objectives?
- Is the architecture designed to accommodate likely changes?
- Does the architecture describe how reused code will be made to conform to other architectural objectives?

3.6. Листа за проверу за архитектуру

3.6.2. General Architectural Quality

• Does the architecture account for all the requirements?
• Is any part over- or under-architected? Are expectations in this area set out explicitly?
• Does the whole architecture hang together conceptually?
• Is the top-level design independent of the machine and language that will be used to implement it?
• Are the motivations for all major decisions provided?
• Are you, as a programmer who will implement the system, comfortable with the architecture?

3.7. Количина времена за предуслове

Количина времена која је потребна за дефиницију проблема, за захтеве и за софтверску архитектуру варира у зависности од потреба вашег пројекта. Уопштено гледано, добро вођени пројекти посвете 10-20% својих напора и око 20-30% свох распореда на захтеве, архитектуру и предуслове. Ове цифре не обухватају детаљни дизајн, зато што детаљни дизајн реализује у оквиру конструкције.

Ако су захтеви нестабилни и ви радите на великом, формалном пројекту, вероватно ћете морати да радите са аналитичарем захтева како би разрешили проблеме захтева који су идентификовани у раној фази конструкције. У том случају, оставите времена за консултацију са аналитичарем захтева како би додатно прегледали захтеве пре добијања верзије захтева погодне за даљи рад.

Ако су захтеви нестабилни и ако радите намањем, неформалном пројекту, обезбедите време за довољно добро дефинисање захтева, тако да њихово евентуално мењање у будућности има минимални утицај на конструкцију.

3.7. Количина времена за предуслове

Ако су у неком пројекту (било формалном или неформалном) захтеви нестабилни, онда се рад на захтевима може третирати као посебни пројекат. Процените време потребно за остатак пројекта тек када завршите захтеве.

Ово је коректан приступ, јер нико разуман не може очекивати да се процени време и распоред пре него што се зна шта се тачно гради. То је слично као да сте предрадник у градњи куће. Кад клијент пита: Колико ће то коштати?, предрадник разумно пита: Шта то треба направити? Ако клијент настави: Не могу одговорити, али ме интересује колико ће ме коштати?, предрадник зна да је даљи разговор беспредметан и да представља трошење његовог времена.

Када се ради о грађевинарству, јасно је да није одговоран онај клијент који очекује да добије понуду пре него што је описао шта треба да се гради. Клијенти неће почети са куповином материјала (цемента, песка, арматуре, цигала, даски, ексера итд.) пре него што је архитекта завршио своје планове. Може се догодити да ваши клијенти не схвате одмах зашто планирате развој захтева као одвојен пројекат, па се јавља потреба да им објасните тај начин размишљања.

3.7. Количина времена за предуслове

Када се алоцира време за архитектуру софтвера, пожељно је користити приступ сличан оном који је коришћен за развој захтева. Ако раније нисте радили са таквом врстом софтвера, обезбедите више времена због несигурности у дизајнирању нове области.

Потребно је да обезбедите да време потребно за креирање добре архитектуре неће бити одузето од времена које вам је потребно за успешан завршетак посла у другим областима.

Ако је неопходно, може се и рад на спецификацији архитектуре такође планирати као одвојени пројекат.

3.7. Количина времена за предуслове

3.7.1. Листа за проверу: предуслови

• Have you identified the kind of software project you're working on and tailored your approach appropriately?
• Are the requirements sufficiently well-defined and stable enough to begin construction (see the requirements checklist for details)?
• Is the architecture sufficiently well defined to begin construction (see the architecture checklist for details)?
• Have other risks unique to your particular project been addressed, such that construction is not exposed to more risk than necessary?

3.8. Рекапитулација

● Најважнији циљ припреме за конструкцију је смањивање ризика. Будите сигурни да ваше припремне активности смањују ризик, а не да га повећавају.

● Ако желите да развијете јако квалитетан софтвер, тада квалитет мора бити у фокусу процеса развоја софтвера од почетка до краја. Фокусирање на квалитет на почетку пројекта има већи утицај на крајњи квалитет производа него фокус ирање на квалитет пред крај пројекта.

● Део посла програмера је да подучи шефове и колеге о процесу развоја софтвера, укључујући важност адекватне припреме пре него што почне програмирање.

● Врста пројекта на ком се ради значајно утиче на предуслове за конструкцију – многи пројекти треба да буду веома итеративни, а неки други треба да буду више секвенцијални.

● Ако није дата добра дефиниција проблема, може вам се догодити да током конструкције решавате погрешан проблем.

3.8. Рекапитулација

● Ако захтеви нису добро урађени, може вам се догодити да промашите важне детаље проблема. Измене у захтевима коштају 20 до 100 пута више ако се реализују у каснијим фазама, па пре почетка програмирања треба бити сигуран да су захтеви прави.

● Ако није направљена добра архитектура, може вам се догодити да током конструкције решавате прави проблем на погрешан начин. Цена измена у архитектури се повећава тиме што се више кода пише за погрешну архитектуру, па стога треба бити сигуран и да је архитектура права.

● Треба разумети који је приступ био коришћен на пројекту за предуслове, па на одговарајући начин изабрати приступ за конструкцију.