Развој софтвера 2

3. Двапут мери, једном сеци: Ране активности - предуслови

3.1. Ране активности - предуслови

Пре почетка градње куће, грађевинар прегледа планове, провери да ли су прибављене све дозволе и прегледа темеље куће. Грађевинар се на један начин припрема за изградњу небодера, на други начин за изградњу породичне куће а на трећи начин за изградњу кућице за псе. Какав год пројекат био у питању, припрема одговара конкретним потребама тог пројекта и врши се савесно пре него што се отпочне са градњом.
Презентација описује активности које се морају реализовати да би се припремили за конструкцију софтвера. Као и приликом градње, успех или неуспех пројекта је великим делом већ одређен пре него што почне изградња. Ако темељи нису добро постављени или планирање није адекватно, највише што се може урадити приликом градње је свођење штете на минимум. Ако желите да направити најсјајнији накит, морате почети од сировог дијаманта. Ако почете са плановима за циглу, најбоље што можете добити је “фенси” цигла.
Изрека “двапут мери, једном сеци” је веома релевантна за део конструкције у развоју софтвера, који може представљати и до 65% укупног трошка пројекта. Најгори софтверски пројекти заврше тако што се конструкција врши два, три или више пута. Ако се најскупљи део посла ради два пута, то није добро ни у развојс софтвера ни у било којој другој производњи.

3.1.1. Значај предуслова

Заједнички фактор за све програмере који производе високо-квалитетан софтвер је њигово коришћење високо-квалитетних пракси. Те праксе стављају акценат на квалитет на почетку пројекта, у средини пројекта и на крају пројекта.
1) Ако је акценат на квалитет на крају пројекта, онда је акценат на тестирање система. Тестирање је оно на шта се помисли када се разматра обезбеђивање квалитета софтвера. Међутим ,тестирање је само један (и то не најутицајнији) део комплетне стратегије за обезбеђење квалитета. Тестирањем се не може детектовати пропуст грађења погрешног производа или грађења правог производа на погрешан начин. Такви пропусти морају да се открију раније, пре него што почне конструкција.
2) Ако је акценат на квалитет у средини пројекта, тада је акценат на праксе конструкције. Такве праксе су у фокусу ове презентације и презентација које следе.
3) Ако је акценат на квалитет на почетку пројекта, тада се праве захтеви, планира и дизајнира високо квалитетан производ. Ако се почне са процесом дизајна аута Југо, он се може тестирати, али се никако не може претворити у Ферари. Можемо се направити најбољи могући Југо, али ако се жели направити Ферари потребно је то планирати од почетка. У развоју софтвера се исто врши слично планирање када се дефинише проблем, када се специфицира решење и када се дизајнира то решење.

3.1.2. Да ли се предуслови примењују на модерне софтверске пројекте?

Неки аутори сматрају да ране активности као што су дефинисање архитектуре, дизајн и планирање пројекта нису од користи у модерним софтверским пројектима. У глобалу гледано, такве тврдње нису подржане ни старијим ни новијим истраживањима, нити са трнутно расположивим подацима. Противници предуслова обично приказују примере где су предуслови били лоше урађени и потом истичу да такав рад није ефикасан. Ране активности се, ипак, могу добро одрадити. Индустријски подаци од 70-тих година прошлог века до данас јасно указују да ће пројекти најбоље напредовати ако се обаве одговарајуће припремне активности пре него што се озбиљно почне са конструкцијом.

Свеобухватни циљ припрема је редуковање ризика: добар планер пројекта што је пре могуће уклони са пута највеће ризике како би се најобимнији делови пројекта глатко одвијали. Најчешћи ризици у развоју софтверских пројеката су лоше креирање захтева и лоше планирање пројектата, па припрема има тенденцију да се фокусира на побољшање захтева и пројектних планова.

Припрема за конструкцију није егзактна наука, па се од пројекта до пројекта разликују одлуке који ће конкретно приступ бити изабран за редукцију ризика. У различитим пројектима се ти подаци увелико разликују.

3.1.3. Разлози за некомплетну припрему

Могло би се сматрати да сви професионални програмери знају важност припреме и да пре преласка у конструкцију проверавају да ли су предуслови задовољени. Нажалост, то често није случај.

Чест узрок некомплетних припрема је да програмери којима је додељен задатак рада на раним активностима немају довољно експертизе за реализацију додељеног задатка. Вештине потребне за планирање пројекта, креирање неодољивих случајева пословања, развој свеобухватних и тачних захтева и за креирање високо квалитетне архитектуре су далеко од тривијалних и највећи број програмера није адекватно утрениран за извршавање тих активности.

Неки програмери знају како да реализују ране активности, али се не припреме зато што се не могу одупрети нагону да почну са кодирањем што је пре могуће.

Још један разлог за неприпремање програмера је то што су менаџери скоро увек немају осећаја за програмере који проводе време на предусловима за конструкцију. Тај феномен је познат под скраћеницом ЗМНКН – Зашто Мика не кодира нешто (енг. WISCA - Why Isn’t Sam Coding Anything).

3.1.3. Разлози за некомплетну припрему

Ако менаџер пројекта нареди да се одмах почне са кодирањем, најлакше је рећи “Разумем, господине” и поступити по наређењу. То у принципу није добар одговор и могле би се размотрити алтернативе:

• Прво, можете глатко одбити да радите на такав, погрешан начин. Ако су ваши односи са сешов и стање вашег рачуна у банци на довољно високом нивоу да можете тако да урадите, срећно вам било.

• Друго, можете се претварати да кодирате чак иако то не радите. Ставите листинг старог програма на ваш радни сто. Онда продужите по вашем плану и направите захтеве и архитектуру, са или без шефовог одобрења. На тај начин ћете брже одрадити пројекат и добити квалитетније резултате. Са тачке гледишта вашег шефа, незнање је благослов.

• Треће, можете едуковати вашег шефа о нијансама код техничких пројеката. То је добар приступ зато што повећава број просветљених шефова на планети.

• На крају, можете да потражите други посао. Упркос развоју и кризама у економији, добрих програмера сво време има јако мало на залихама, а живот је сувише кратак да би радили у непросвећеном окружењу када се већи број бољих алтернатива налази на располагању.

3.1.4. Чврсти аргументи за рад на предусловима пре конструкције

Део вашег посла, као технички образованог запосленог (енг. technical employee – techiee) је образовање људи око вас, који нису технички образовани, да разумеју процес развоја софтвера. Ово ће вам помоћи да се изборите са менаџером и шефовима који "још нису угледали светло". То је листа аргумената за промовисање праксе креирања захтева и архитектуре (тј. обезбеђење критичних аспеката) пре него што се почне са кодирањем, тестирањем и дебагирањем.

Апеловање на логику

Једна од кључних идеја код ефикасног програмирања је важност припреме. Има смисла да се, пре него што почне са радом на великом пројекту, треба планирати. Велики пројекти захевају више планирања, а мањи пројекти захтевају мање планирања. Са менаџерске тачке гледишта, планирање значи одређивање количине времена, броја људи и броја рачунара који су потребни за пројекат. Са техничке тачке гледишта, планирање значи разумевање шта се жели изградити тако да се не потроше средства за градњу погрешног производа. Понекад корисници на почетку нису потпуно сигурни шта тачно желе, па ће можда требати више напора него што би било идеално како би се одредило шта им стварно треба. Али и то је јефтиније од прављења погрешног производа, његовог одбацивања и поновног почетка.

3.1.4. Чврсти аргументи за рад на предусловима пре конструкције

Апеловање на аналогију

Изградња софтверског система је као било који други пројекат који захтева људе и новац. Ако градите кућу, прво се направе архитектонски цртежи и планови, па се тек онда почне са укуцавањем ексера. Ваши планови бивају прегледани и одобрени пре него шсто поченте да сипате бетон. И код софтвера је исто тако потребно постојање техничког плана.

Не почињете са украшавањем јелке пре него што се та јелка усправи и учврсти. Не палите ватру пре него што сте подложили и наслагали дрва. Не крећете на дугачак пут са аутом у коме је ниво горива на резерви. Не облачите се пре туширања, нити прво обучете ципеле, па тек онда чарапе (изузетак за ово је случај јаке поледице). И код софтвера се ствари морају радити у правом редоследу.

Програмери су на крају ланца исхране. Софтверске архитекте конзумирају захтеве, дизајнери конзумирају архитектуру, а кодери конзумирају дизајн.

3.1.4. Чврсти аргументи за рад на предусловима пре конструкције

Упоредимо софтверски ланац исхране са реалним ланцем исхране. У еколошком окружењу морски галебови једу свежу туну. То је храњиво за галебове, зато што туне једу свеже харинге, које даље једу свеже планктоне. Резултат је здрав ланац исхране. У програмерском ланцу исхране, ако је здрава храна на сваком беочугу ланца, тада ће резултат бити здрав код који су написали срећни програмери.

У загађеном окружењу, планктони су пливали у радиоактивном отпаду. Харинге бивају заражене, па туне које једу харинге такође постају болесне. Код морских галебова, који се налазе на рају ланца исхране, концентрација загађења постаје опасна за њихов живот. У програмерском ланцу исхране, ако су захтеви контаминирани, тада они контаминирају архитектуру, која надаље контаминира конструкцију. Ово доводи до пргавих и прегладнелих програмера и до радиоактивног тј. загађеног софтвера који је изрешетан дефектима тј. грешакама.

3.1.4. Чврсти аргументи за рад на предусловима пре конструкције

Ако планирате веома итеративан пројекат, биће потребно да се идентификују критични захтеви и архитектонски елементи који ће се применти на сваки део конструисаног – и то пре него што се отпочне са конструкцијом. Градитељ групе кућа не мора пре градње прве куће да зна сваки детаљ о свакој кући коју треба изградити. Али он треба да прегледа локацију, да означи трасу за струју, воду и канализацију итд. Ако се градитељ не припреми адекватно, онда може доћи до кашњења градње када се буде копала канализација испод куће која је већ изграђена.

Апеловање на податке

Студије у протеклих 25 година су показале да се исплати када се праве ствари раде благовремено. Веома су скупе измене које нису неопходне. истраживачи у Hewlett-Packard, IBM, Hughes Aircraft, TRW и другим организацијама су открили да отклањање грешке на почетку конструкције допушта поправку која је 10 до 100 пута јефтинија него онда када се исправља у последњем делу процеса, тј. током теста система или након испоруке.

3.1.4. Чврсти аргументи за рад на предусловима пре конструкције

Уопштено говорећи, принцип је да се грешка открије у времену које је што ближе времену њеног настанка. Како се прво креирају захтеви, то грешке у захтевима имају потенцијал да дуже буду у системима и стога су оне скупље. Грешке које су рано унесене у систем имају тенденцију да испољавају шире ефекте од грешака које се креирају касније. Стога су и ране грешке много скупље.

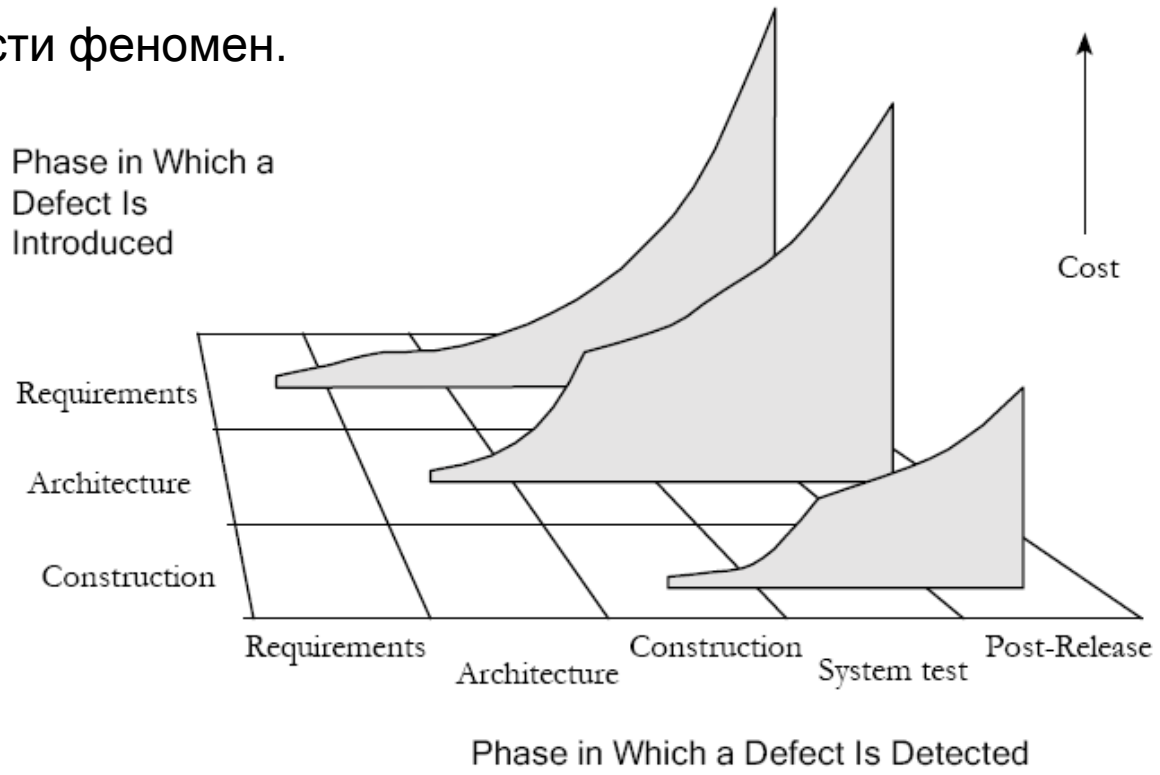
Табела 3-1. *Просечна цена поправке грешке у зависности од тога кад је настала и кад је откривена*

	Time Detected				
Time Introduced	Re-quire-ments	Archi-tecture	Con-struc-tion	System Test	Post-Re-lease
Requirements	1	3	5-10	10	10-100
Architecture	—	1	10	15	25-100
Construction	—	—	1	10	10-25

Подаци из Табеле Table 3-1 показују да, на пример, да је цена исправке грешке настале при креирању архитектуре $1000 када се исправља током креирања архитектуре и $15 000 када се исправља током теста система.

3.1.4. Чврсти аргументи за рад на предусловима пре конструкције

Слика 3-1 илуструје исти феномен.



Слика 3-1
Трошкови отклањања грешака

Просечан пројекат ипак испољава највећи део активности на корекцији грешака у десном крају Слике 3-1, што значи да дебагирање и поновни рад који му је придружен заузимају око 50% времена у типичном циклусу развоја софтвера. На десетине компанија су схватиле да се једноставним фокусирањем на раније исправљање проблема у реализацији пројекта могу умањити и тршкови и време потребно за развој софтвера и то за фактор два или више.

3.1.5. Одређивање какав се софтвер развија

Каспер Џонс је сумирао 20 година истраживања софтвера закључком да су његове колеге и он гледали 40 различитих методологија прикупљања захтева, 50 варијација у раду на дизајну софтвера и 30 врста тестирања које су се примењивале на пројекте реализоване помоћу више од 700 различитих програмских језика.

Различите врсте софтверских пројеката захтевају различите балансе између припреме и конструкције. Сваки пројекат је јединствен, али пројекти имају тенденцију да се нађу унутар неког од општих стилова програмирања.

Табела 3-2 *Добре праксе за три уобичајене врсте софтверских пројеката*

	Typical Good Practices		
Kind of Software	**Business Systems**	**Mission-Critical Systems**	**Embedded Life-Critical Systems**
Typical applications	Internet site Intranet site Inventory management Games Management information systems Payroll system	Embedded software Games Internet site Packaged software Software tools Web services	Avionics software Embedded software Medical devices Operating systems Packaged software

3.1.5. Одређивање какав се софтвер развија

Табела 3-2 *Добре праксе за три уобичајене врсте софтверских пројеката*

	Typical Good Practices		
Kind of Software	**Business Systems**	**Mission-Critical Systems**	**Embedded Life-Critical Systems**
Lifecycle models	Agile development (extreme programming, scrum, timebox development, and so on) Evolutionary prototyping	Staged delivery Evolutionary delivery Spiral development	Staged delivery Spiral development Evolutionary delivery
Planning and management	Incremental project planning As-needed test and QA planning Informal change control	Basic up-front planning Basic test planning As-needed QA planning Formal change control	Extensive up-front planning Extensive test planning Extensive QA planning Rigorous change control

3.1.5. Одређивање какав се софтвер развија

Табела 3-2 *Добре праксе за три уобичајене врсте софтверских пројеката*

	Typical Good Practices		
Kind of Software	**Business Systems**	**Mission-Critical Systems**	**Embedded Life-Critical Systems**
Require-ments	Informal require-ments specification	Semi-formal re-quirements specifica-tion As-needed require-ments reviews	Formal requirements specification Formal requirements inspections
Design	Design and coding are combined	Architectural design Informal detailed design As-needed design reviews	Architectural design Formal architecture inspections Formal detailed de-sign Formal detailed de-sign inspections

3.1.5. Одређивање какав се софтвер развија

Табела 3-2 *Добре праксе за три уобичајене врсте софтверских пројеката*

	Typical Good Practices		
Kind of Software	**Business Systems**	**Mission-Critical Systems**	**Embedded Life-Critical Systems**
Construction	Pair programming or individual coding Informal check-in procedure or no check-in procedure	Pair programming or individual coding Informal check-in procedure As-needed code reviews	Pair programming or individual coding Formal check-in procedure Formal code inspections
Testing and QA	Developers test their own code Test-first development Little or no testing by a separate test group	Developers test their own code Test-first development Separate testing group	Developers test their own code Test-first development Separate testing group Separate QA group
Deployment	Informal deployment procedure	Formal deployment procedure	Formal deployment procedure

3.1.5. Одређивање какав се софтвер развија

Пројекти за пословне системе имају тенденцију да користе високо итеративне приступе, у којима су планирање, захтеви и архитектура испреплетени са конструкцијом, тестирањем система и активностима на обезбеђењу квалитета. Системи за одржавање живота имају тенденцију да захтевају секвенцијалнији приступ и код њих је стабилност захтева један неопходан услов за обезбеђење изузетно високог нивоа поузданости.

Неки аутори су тврдили да пројекти који користе итеративне технике уопште не морају да се много коннцентришу на предуслове, али тај поглед је превише поједностављен. Итеративни приступи имају тенденцију умањења негативних утитицаја неадекватно реализованих раних активности, али не могу елиминисати тај негативни утицај у потпуности.

Размотримо приступ број 1, описан у одговарајућој колони Табеле 3-3. Ту се ради о пројекту који се реализује секвенцијално и који се искључиво ослања на тестирање ради откривања проблема. У оваквом приступу, исправљање проблема, тј. поновни рад, ће бити груписан на крају пројекта.

3.1.5. Одређивање какав се софтвер развија

Табела 3-3 *Приступи у развоју пројекта*

	Approach #1		Approach #2	
	Sequential Approach without Prerequisites		Iterative Approach without Prerequisites	
Project completion status	Cost of Work	Cost of Rework	Cost of Work	Cost of Rework
10%	$100,000	$0	$100,000	$75,000
20%	$100,000	$0	$100,000	$75,000
30%	$100,000	$0	$100,000	$75,000
40%	$100,000	$0	$100,000	$75,000
50%	$100,000	$0	$100,000	$75,000
60%	$100,000	$0	$100,000	$75,000
70%	$100,000	$0	$100,000	$75,000
80%	$100,000	$0	$100,000	$75,000
90%	$100,000	$0	$100,000	$75,000
100%	$100,000	$0	$100,000	$75,000
End-of-Project Rework	$0	$1,000,000	$0	$0
TOTAL	$1,000,000	$1,000,000	$1,000,000	$750,000
GRAND TOTAL		$2,000,000		$1,750,000

Подаци само служе у сврху илустрације

3.1.5. Одређивање какав се софтвер развија

Итеративни пројекат који скраћује или елиминише предуслове ће се на два начина разликовати од секвенцијалног пројекта који се на исти такав начин односи према предусловима. Прво, просечна цена поправљања грешке ће бити нижа, јер ће постојати тенденција откривања грешки ближе времену њиховог настанка. Ипак, грешке ће и даље бити откриване у каснијој фази итерације, па ће њихово поправљање захтевати да делови софтвера буду редизајнирани, поново кодирани и поново тестирани, што цену исправке грешке чини већом него што би могла да буде.

Друго, код итеративног приступа се цена поновног рада апсорбује део по део током пројекта, а не бива груписана на самом крају пројекта. Када се, на крају, слегне сва прашина, може се видети да су укупни трошкови приближно једнаки. Међутим, у итеративном приступу се ти трошкови плаћају у ратама током рада на пројекту, док се у секвенцијалном приступу ти трошкови плаћају одједном. Као што и приказује Табела 3-4, фокусирање на предуслове може умањити укупне трошкове, без обзира да ли се користи итеративни или секвенцијални приступ. Из много разлога су итеративни приступи чешће боља опција, али треба истаћи да итеративни приступ који игнорише предуслове може на крају коштати знатно више него секвенцијални приступ у коме се предусловима поклања адекватна пажња.

3.1.5. Одређивање какав се софтвер развија

Табела 3-4 *Приступи у развоју пројекта*

	Approach #3		Approach #4	
	Sequential Approach with Prerequisites		Iterative Approach with Prerequisites	
Project completion status	Cost of Work	Cost of Rework	Cost of Work	Cost of Rework
10%	$100,000	$20,000	$100,000	$10,000
20%	$100,000	$20,000	$100,000	$10,000
30%	$100,000	$20,000	$100,000	$10,000
40%	$100,000	$20,000	$100,000	$10,000
50%	$100,000	$20,000	$100,000	$10,000
60%	$100,000	$20,000	$100,000	$10,000
70%	$100,000	$20,000	$100,000	$10,000
80%	$100,000	$20,000	$100,000	$10,000
90%	$100,000	$20,000	$100,000	$10,000
100%	$100,000	$20,000	$100,000	$10,000
End-of-Project Rework	$0	$0	$0	$0
TOTAL	$1,000,000	$200,000	$1,000,000	$100,000
GRAND TOTAL		$1,200,000		$1,100,000

Подаци само служе у сврху илустрације

3.1.5. Одређивање какав се софтвер развија

Као што и сугерише Табела 3-4, највећи број пројеката није ни потпуно секвенцијалан ни потпуно итративан. Није практично да се унапред специфицира 100% захтева или 100% дизајна, али највећи део пројеката налази вредност у раном идентификовању бар оних најкритичнијих захтева и најкритичнијих елемената архитектуре.

Један од реалистичних приступа је план да се рано, тј. Унапред, специфицира око 80% захтева, да се алоцира време за касније специфицирање додатних захтева и да се потом практикује систематска контрола промена и на тај начин се обезбеди да током рада на пројекту буду прихваћени само највреднији међу новим захтевима.

3.1.5. Одређивање какав се софтвер развија

Одлука до које мере предуслови треба да буду унапред дефинисани ће варирати у зависности од типа пројекта (из Табеле 3-2), од формалности пројекта, од техничког окружења, од способности запослених и од пословних циљева пројекта.

Секвенцијални (тј. унапред дефинисан) приступ се може користити када:

- Захтеви су веома стабилни
- Дизајн је директан и веома добро схваћен
- Развојни тим је фамилијаран са пословном логиком апликације
- Пројекат садржи мало ризика
- Важна је дугорочна предвидивост
- Очекује се да буде висока цена измене захтева, дизајна и кода током рада на пројекту

3.1.5. Одређивање какав се софтвер развија

Итеративнији приступ се може користити када:

- Захтеви нису добро схваћени или се због неких других разлога очекује да неће бити стабилни
- Дизајн је превише сложен, превише изазован, или и једно и друго
- Развојни тим није фамилијаран са пословном логиком апликације
- Пројекат садржи много ризика
- Није важна дугорочна предвидивост
- Очекује се да буде ниска цена измене захтева, дизајна и кода током рада на пројекту

3.1.5. Одређивање какав се софтвер развија

Можете да прилагодите предуслове конкретноим пројекту тако што ћете те предуслове начинити мање или више формалним или мање или више комплетним, како вам више одговара.

Утицај на конструкцију се огледа у томе што прво треба одредити каоји су предуслови за конструкцију одговарајући за ваш пројект. Неки пројекти проведу премало времена на предусловима, чиме се конструкција излаже непотребно великом ризику дестабилирајућих промена и тиме се спречава да пројекат има конзистентан напредак. У неким пројектима има превише раних активности. Такви пројекти се упорно придржавају затева и планова која су даља открића приликом рада учинила инвалидним, што такође може угрозити напредак током конструкције.

3.2. Предуслови: Дефиниција проблема

Први предуслов који се треба испунити пре почетка конструкције је јасна изјава о проблему који би систем требао да решава. То се понегде назива “визија производа” (енг. “product vision”), “изјава о мисији” (енг. “mission statement”) или “дефиниција производа” (енг. “product definition”). Овде ће тај докуменат бити назван дефиниција проблема (енг. “problem definition”).

Дефиниција проблема дефинише шта је то проблем а да притом не реферише на ниједно од могућих решења. То је једноставан документ,величине једна до две стране и он треба да звучи као проблем. Изјава “Не можемо да ажурно пратимо поруџбине од Техноманије” звучи као проблем и представља добру дефиницију проблема. Изјава “Потребно је да оптимизујемо наш систем уноса података како би ажурно пратили поруџбине од Техноманије” је лоша дефиниција проблема. Она не звучи као проблем, већ као решење.

Дефиниција проблема претходи детаљном раду на захтвима, који представља дубље истраживање проблема.

Дефиниција проблема треба да буде исказана у језику корисника и проблем треба да буде описан са тачке гледишта корисника.

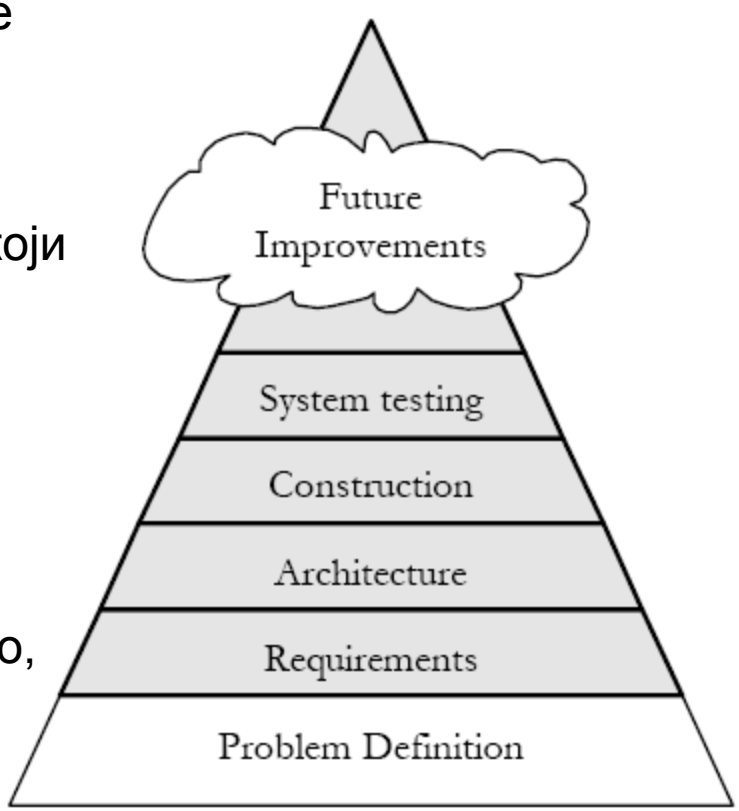
3.2. Предуслови: Дефиниција проблема

Дефиниција програма обично не треба да буде исказана у техничким терминима рачунарства. Најбоље решење чак не мора ни да буде рачунарски програм.

Претпоставимо да треба направити извештај који приказује годишњи профит и да постојећи софтвер већ садржи извештаје који приказују кварталне профите.

Ако сте закључани у програмерским размишљањима, ваш резон ће бити да је додавање годишњег извештаја у систем који садржи кварталне извештаје једноставан посао, па ћете платити програмера да напише и дебагира програм за рачунање годишњег профита.

Ако нисте закључани у рачунарским размишљањима, онда ћете наредити секретарици да срачуна годишње цифре тако што ће сабрати бројке из кварталних извештаја.



Слика 3-2 *Фазе у пројекту*

3.2. Предуслови: Дефиниција проблема

Слика 3-3 *Ако не постоји добра дефиниција проблема, може се догодити да се уложи напор у решавање погрешног проблема. Будите сигурни да сте добро нацињали пре него што пуцате.*

Казна за неуспех у дефинисању проблема може бити да сте потрошили много времена у решавању погрешног проблема. Ту у ствари постоји двострука казна, јер не само што је узалуд потрошено време и труд, већ је све то време прави проблем није решаван.

3.3. Предуслови: Захтеви

Захтеви детаљно описују шта треба да ради софтверски систем и они представљају први корак према решењу. Активност око захтева се још назива и развој захтева, анализа захтева, анализа, дефинисање захтева, софтверски захтеви, спецификација, функционална спецификација.

Зашто треба имати званичне захтеве?

Експлицитан скуп захтева је важан из више разлога:

•Експлицитни захтеви помажу да се обезбеди да корисник, а не програмер, води функционалност система. Ако су захтеви експлицитни, тада их корисник може прегледати и сложити се са њима. Ако захтеви нису експлицитни, тада програмер обично ради тако што одлуке о захтевима доноси током програмирања.

•Експлицитни захтеви помажу да се програмер заштити од погађања шта то корисник жели.

•Експлицитни захтеви такође помажу у избегавању расправа и свађа.

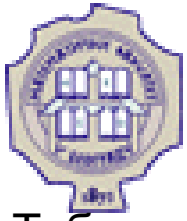
Одлуке о опсегу обухвата софтверског система се доносе пре почетка програмирања. Ако постоји неслагање између програмера око тога шта програм треба да ради, они се обично могу разрешити прегледом писаних захтева.

3.3. Предуслови: Захтеви

Давање одговарајуће пажње захтевима помаже да се минимизирају промене у систему по почетку развоја:

•Ако се током кодирања открије грешка у кодирању, потребно је променити неколико линија и рад се наставља.

•Ако се током кодирања нађе грешка у захтевима, тада се мора мењати дизајн како би се изашло у сусрет промењеним захтевима. Дакле, биће потребно да се одбаци стари дизајн. Будући да се нови дизајн треба прилагодити коду који је већ написан, то ново дизајнирање ће трајати дуже него први пут. Надаље, неопходно је одбацити код и случајеве тестирања који су погођени променом захтева, па поново написати и једно и друго. Чак и код који није погођен променама треба да буде поново тестиран, како би се било сигурно да промене у другим областима нису довеле до нових грешака.



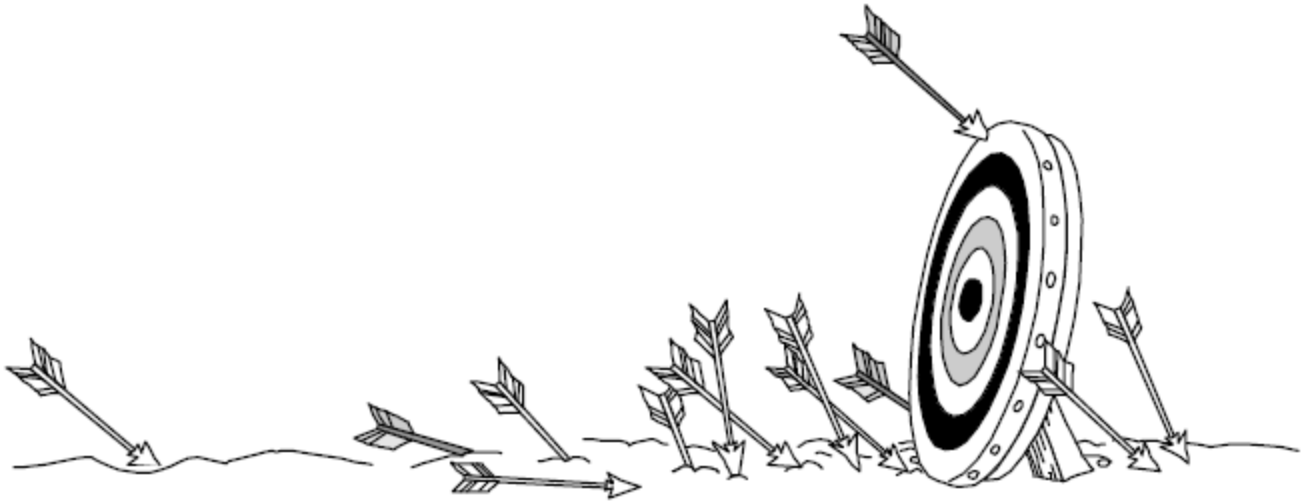
3.3. Предуслови: Захтеви

Табела 3-1 указује, а подаци из великог броја анализа потврђују, да је у великим пројектима грешка захтева откривена током креирања архитектуре око 3 пута скупља за поправку него што би била да је уочена приликом креирања захтева. Ако је иста грешка откривена током кодирања, она је скупља 5-10 пута. Ако је таква грешка откривена током тестирања система, скупља је око 10 пута. Уколико је грешка у захтевима откривена после испоруке, биће скупља 10-100 пута, при чему на мањим пројектима, где су мањи административни трошкови, фактор пре варира између 5 и 10 него 10 и 100. Колико год била та повећана цена, сигурно је да не бисте били срећни ако се тај новац одбија од ваше плате.

Табела 3-1. *Просечна цена поправке грешке у зависности од тога кад је настала и кад је откривена*

	Time Detected				
Time Introduced	Re-quire-ments	Archi-tecture	Con-struc-tion	System Test	Post-Re-lease
Requirements	1	3	5-10	10	10-100
Architecture	—	1	10	15	25-100
Construction	—	—	1	10	10-25

3.3. Предуслови: Захтеви



Слика 3-4
Ако нема добрих захтева, може се догодити да се гађа прави проблем, али да се промашује мета која се односи на специфичне аспекте тог проблема.

Дакле, специфицирање захтева је кључно за успех пројекта, чак и у већој мери него што је то случај са избором адекватних и ефикасних техника консруисања.

3.3. Предуслови: Захтеви

Захтеви су као вода. Најлакше је градити на њој када је замрзнута.

Мит о стабилним захтевима

Стабилни захтеви су “свети грал” у развоју софтвера. Када постоје стабилни захтеви, пројекат може напредовати од архитектуре, преко дизајна до кодирања и тестирања на уређен смирен и предвидив начин. Тиме је достигнут “софверски рај”! Трошкови ће бити предвидиви и неће морати да се брине о карактеристици која постаје сто пута скупља за имплементацију зато што корисник о њој није размишљао све док тренутка завршетка дебагирања.

Лепо је надати се да, једном када клијент прихвати документ са захтевима, неће бити потребе за изменама. Међутим, у типичном пројекту, клијент не може поуздано да опише шта му је потребно пре него што је код написан. Проблем није у томе да су клијенти нижа животна форма, већ се ради о томе да, што више радите са пројектом, тај пројекат боље разумете, па ако још више радите, онда ћете га још боље разумети. Дакле, процес развоја софтвера помаже клијентима да боље схвате своје потребе, што је главни извор захтева. План да се ригидно прати документ са захтевима је план да се не одговара клијенту, што је обично неприхватљиво.

Колико је промена уобичајено? У студијама IBM-а и других компанија је закључено да просечан пројекат има око 25% промена у захтевима насталих током развоја, што представља 70-85% укупног поновног рада.

3.3. Предуслови: Захтеви

Можда сматрате да је “југић” најбољи ауто који је икад направљен, да је Земља равна плоча или да ће фудбалски тим Хајдука из Куле за две године освојити Лигу Шампиона. Ако је тако, наставите у том правцу, па можете сматрати да се захтеви у пројектима на којима радите неће мењати. Ако сте престали да верујете у Деда Мраза, Зубић Вилу или Божић Бату, или бар престали да признајете да верујете у њих, тада можете предузети неке кораке ради минимизације утицаја промена у захтевима.

Руковање са променама у захтевима током конструкције

Следи списак акција које се могу предузети да би се најбоље руковало променама у захтевима током конструкције:

Користити листу за проверу за захтеве, дату на крају ове секције, како би се контролисао квалитет документа са захтевима

Ако захтеви нису довољно добри, прекините са радом, вратите се на захтеве, учините да буду добри и тек онда наставите. Јесте да изгледа као да касните ако у том тренутку зауставите кодирање, али ако се возите од Београда према Новом Саду, да ли је губљење времена ако станете и погледате мапу када видите таблу са натписом Пожаревац? Naравно да није. Ако се нисте упутили у добром правцу, паметно је зауставити се и проверити позицију.

3.3. Предуслови: Захтеви

Обезбедити да сви буду упознати са ценом промене захтева
Приликом размишљања о новим карактеристикама софтвера који се развија, клијенти постану узбуђени. У свом узбуђењу, они добију вртоглавицу и забораве све пређашње састанке на којима су се разматрали захтеви, церемонију потписивања захтева, као и комплетан документ са захтевима. Најлакши начин комуникације са људима који су “отровани” новим карактеристикама је, отприлике, овакав: “Ех, то је баш добра идеја. Како је нема у захтевима, ја ћу урадити промене у календару рада, као и нову процену трошкова, како би могли јасно да одредимо да ли ћемо ту идеју реализовати сада или касније”. Речи “календар рада” и “трошкови” боље отрезне него кафа и хладан туш, па се многе “мора постојати” карактеристике брзо преобрате у “пожељно да постоји”.

Ако организација у којој радите не увиђа важност потребе да се прво направе захтеви, може се истаћи како су промене у захтевима много јефтиније од каснијих промена.

3.3. Предуслови: Захтеви

Подесити процедуру за контролу промена

Ако клијентово узбуђење и даље траје, може се установити формални одбор за контролу промена, са задатком да разматрања предложених промена. У реду је да се клијенти предомисле и да схвате да су им потребне додатне могућности. Проблем је што се њихови предлози тако често мењају да се то не може испратити. Постојање уграђене процедуре за контролу промена чини да сви буду срећни. Ви сте срећни зато што ћете морати да се суочавате са променама само у одређеним тренутцима. Клијенти су срећни зато што виде да постоји план обраде и одговора на њихове сугестије.

Користити приступе у развоју софтвера који се прилагођавају променама

Неки приступи у развоју максимизирају способност одговра на промене у захтевима. Приступ еволутивног прототипа помаже у истраживању системских захтева пре него што снаге крену у изградњу. Еволутивна испорука је приступ по ком се систем испоручује у фазама. Мало се гради, мало се добије повратна информација од корисника, мало се прилагоди дизајн, направи неколико промена, поново мало гради итд. Кључ у том приступу је коришћење кратких циклуса развоја, тако да се брзо може одговорити на захтеве корисника.

3.3. Предуслови: Захтеви

Одбацити пројекат

Ако су захтеви исувише лоши или сувише променљиви и ако ниједан од ранијих прелога не даје резултате, откажите пројекат.

Чак иако не можете заиста да откажете пројекат, размотрите шта би било када би га отказали. Размотрите колико још горе треба да буде, тј. Који праг треба да се пређе, па да одлучите да заиста откажете пројекат.

Ако већ знате за случајеве отказаних пројеката, бар се запитајте колико се ти случајеви разликују од вашег случаја.

3.3. Предуслови: Захтеви

Листа за проверу за захтеве

Листа за проверу за захтеве садржи листу питања које треба поставити о документу са захтевима пројекта. Презентација не описује како развити добре захтеве, па ни листа не одговара на то питање. Она само помаже у провери да ли су развијени захтеви добри. Листа која следи треба да послужи као разумна провера у времену конструкције, ради утврђивања чврстине тла на коме се стоји, тј. где се захтеви налазе “на Рихтеровој скали”.

Неће баш све листе за проверу бити примењиве на дати пројекат. Ако се ради о неформалном пројекту, тада се о неким питањима из листе провера чак ни не размишља.

Постојаће и друга питања о којима би требало размислити, која нису укључена у лсите за проверу. Ако се ради на великим, формалним пројектима, онда се писмено треба одговорити на свако такво питање.

3.3. Предуслови: Захтеви

Specific Functional Requirements

- Are all the inputs to the system specified, including their source, accuracy, range of values, and frequency?
- Are all the outputs from the system specified, including their destination, accuracy, range of values, frequency, and format?
- Are all output formats specified for web pages, reports, and so on?
- Are all the external hardware and software interfaces specified?
- Are all the external communication interfaces specified, including handshaking, error-checking, and communication protocols?
- Are all the tasks the user wants to perform specified?
- Is the data used in each task and the data resulting from each task specified?

3.3. Предуслови: Захтеви

Specific Non-Functional (Quality) Requirements

- Is the expected response time, from the user's point of view, specified for all necessary operations?
- Are other timing considerations specified, such as processing time, data transfer rate, and system throughput?
- Is the level of security specified?
- Is the reliability specified, including the consequences of software failure, the vital information that needs to be protected from failure, and the strategy for error detection and recovery?
- Is maximum memory specified?
- Is the maximum storage specified?
- Is the maintainability of the system specified, including its ability to adapt to changes in specific functionality, changes in the operating environment, and changes in its interfaces with other software?
- Is the definition of success included? Of failure?

3.3. Предуслови: Захтеви

Requirements Quality

- Are the requirements written in the user's language? Do the users think so?
- Does each requirement avoid conflicts with other requirements?
- Are acceptable trade-offs between competing attributes specified - for example, between robustness and correctness?
- Do the requirements avoid specifying the design?
- Are the requirements at a fairly consistent level of detail? Should any requirement be specified in more detail? Should any requirement be specified in less detail?
- Are the requirements clear enough to be turned over to an independent group for construction and still be understood?
- Is each item relevant to the problem and its solution? Can each item be traced to its origin in the problem environment?
- Is each requirement testable? Will it be possible for independent testing to determine whether each requirement has been satisfied?
- Are all possible changes to the requirements specified, including the likelihood of each change?

3.3. Предуслови: Захтеви

Requirements Completeness

- Where information isn't available before development begins, are the areas of incompleteness specified?
- Are the requirements complete in the sense that if the product satisfies every requirement, it will be acceptable?
- Are you comfortable with all the requirements? Have you eliminated requirements that are impossible to implement and included just to appease your customer or your boss?