SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Sustavi za praćenje i vođenje procesa

Proces razvoja softvera

Ivo Veseli

0036396967

Zagreb, svibanj 2007.

Sadržaj

1. Uvod

U razvoj softverskog proizvoda nametnula se struktura koja se zove proces razvoja softvera. Sinonimi ovog procesa uključuju „životni ciklus softvera“ i „softverski proces“. Postoji nekoliko modela za te procese, a svaki opisuje pristupe različitim zadatcima i aktivnostima koje nastaju tijekom procesa.

1.1 Procesi i meta-procesi

Sve više organizacija koje se bave proizvodnjom softvera uvode procesne metode razvoja softvera. Dosta takvih organizacija rade za vojnu industriju, koja zahtjeva ocjenjivanje bazirano na procesnim modelima kao uvjet za dobivanje ugovora. Internacionalni standard koji se koristi za opisivanje metode selekcije, implementacije, i motrenja životnog ciklusa softvera je ISO 12207. CMM (Capability Maturity Model) je jedan od vodećih modela. Neovisni procjenitelji ocjenjuju organizacije na temelju toga kako ta organizacija slijedi svoje definirane procese, a ne na temelju kvalitete tih procesa ili proizvedenog softvera. CMM je postupno zamijenjen sa CMMI. ISO 9000 opisuje standarde za formalno organizirane procese sa dokumentacijom.

ISO 15504, poznat kao SPICE (Software process improvement capability determination) je okvir za procjenu softverskih procesa. Cilj ovog standarda je postavljanje jasnih modela za procese usporedbe. SPICE ima sličnu primjenu kao i CMM, te CMMI. Ti modeli procesiraju upravljanje, kontrolu, vođenje i monitoring softverskog razvoja. SPICE

model se koristi i za ocjenjivanje samog projektnog tima koji radi na razvoju softvera. Te informacije se analiziraju da bi se identificirale slabosti i uočili elementi za poboljšanja.

1.2 Faze u razvoju softvera

Razvoj softvera možemo podijeliti u nekoliko faza: analiza elemenata softvera, specifikacija, arhitektura, implementacija, testiranje, izrada dokumentacije i održavanje.

1.2.1 Analiza elemenata softvera

Najvažniji zadatak u kreiranju softverskih proizvoda je ekstahiranje zahtjeva. Klijent obično zna što želi, ali ne zna kako softver treba raditi. Analizu klijentovih zahtjeva koji su nedovršeni, dvosmisleni i često kontradiktorni, vrše iskusni softverski inženjeri. Oni zatim rade testne verzije softvera i takve verzije prezentiraju klijentu u svrhu smanjenja rizika od loše definiranih zahtjeva.

1.2.2 Specifikacija

Specifikacija je faza u kojoj se precizno definiraju karakteristike koje softver mora zadovoljavati. U praksi je većina kvalitetnih specifikacije napisana tako da isprofilira već gotove aplikacije, prema zahtjevima koji su nam potrebni.

1.2.3 Arhitektura softvera

Arhitektura softverskog sustava odnosi se na apstraktnu reprezentaciju tog sustava. Arhitektura se brine da softverski sustav zadovolji zahtjeve proizvoda, ali isto tako osigura da budući zahtjevi mogu biti riješeni nadogradnjom. Arhitektura također omogućuje povezivanje između softverskog sustava i ostalih softverskih proizvoda (npr. operacijskog sustava).

1.2.4 Testiranje

Testiranje je važan dio izrade kvalitetnog softvera. U ovom dijelu softverski inženjeri pokušavaju uočiti i ukloniti greške koje su nastale kod implementacije, te općenito optimiziraju softverski proizvod.

1.2.5 Izrada dokumentacije

Važan (često i propušten) dio je izrada dokumentacije dizajniranog softvera u svrhu budućeg održavanja i nadogradnje. Dokumentacija mora biti pisana jezgrovito, jasno i precizno kako bi se maksimalno olakšao posao održavanja softvera drugim inženjerima koji će za to biti odgovorni.

1.2.6 Podrška i osposobljavanje

Razlog velikog postotka neuspjelih projekata je u tome što razvojni tim ne uspije shvatiti da nije bitno koliko vremena i planiranja se utroši u planiranje softvera ako ga nitko poslije toga ne koristi. Ljudi ponekad nisu skloni promjenama i pružaju otpor prema novim verzijama softvera, te je zato veoma važno imati kvalitetne korisničke obuke.

1.2.7 Održavanje

Da bi se mogli uspješno boriti sa novootkrivenim problemima, u održavanje i nadogradnju treba uložiti mnogo više vremena nego na inicijalni razvoj softvera. Ne samo da će biti neophodno napisati dodatni kod koje ne odgovara originalnom dizajnu, već će trebati determinirati kako radi softver, što će značiti dodatan napor za softverske inženjere. Otprilike 2/3 softverskih inženjera radi na održavanju, ali samo mali dio na otkrivanju i ispravljanju *bugova*. Većina održavanja sastoji se od proširenja sustava za nove stvari. Za usporedbu, oko 2/3 arhitekata, inženjera i građevinara radi na sličan način na održavanju.

2. Procesni modeli

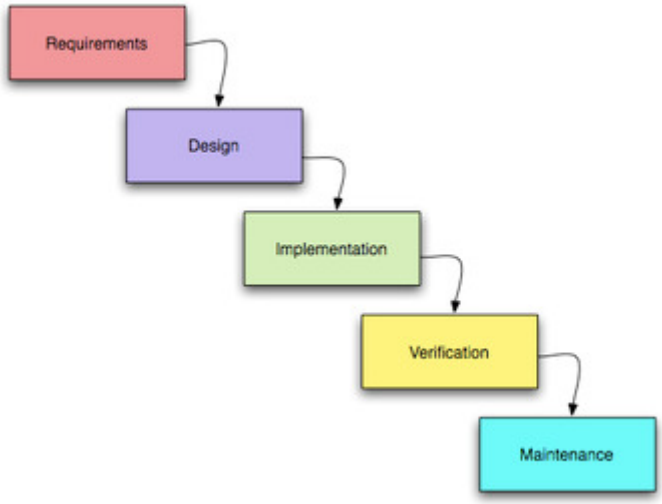
Dugogodišnji cilj softverskog inženjerstva je bio pronaći ponovljive, predvidljive procese ili metodologije koji poboljšavaju produktivnost i kvalitetu softvera. Neki inženjeri pokušavaju sistematizirati ili formalizirati naizgled samovoljan zadatak pisanja softvera. Drugi pak pokušavaju primijeniti tehnike vođenja projekata u pisanju softvera. Bez vođenja projekta, softverski projekti lako mogu biti zakašnjeli ili izvan budžeta. S obzirom da većina softverskih projekata nije zadovoljila očekivanja u smislu funkcionalnosti, troškova i vremena isporuke, uspješno vođenje softverskog projekta je dosta zahtjevno. U nastavku će biti dan pregled nekoliko glavnih modela za procese razvoja softvera.

2. 1 Waterfall model

Waterfall model je sekvencijalni model softverskog razvoja u kojem je razvoj viđen kao stabilni padajući tok (poput vodopada) kroz faze: analiza zahtjeva, dizajn, implementacija, testiranje, integracija i održavanje. Porijeklo izraza „Waterfall“ se povezuje s člankom W. W. Roycea koji je objavljen 1970.

2.1.2 Povijest Waterfall modela

1970. g. W. W. Royce je predstavio krnji model, za kojeg je tvrdio da se može razviti u iterativni, a danas se odnosi na Waterfall model. Unatoč Royceovim namjerama da Waterfall model bude modificiran u iterativni, upotreba Waterfall modela kao čisto sekvencijalnog procesa je i dalje popularna.



Sl. 1. Slika prikazuje faze Waterfall modela

Da bi slijedili Waterfall model moramo napredovati od jedne faze prema drugoj u čisto sekvencijalnom stilu. Npr. prvo trebamo napraviti specifikaciju zahtjeva, što čini kamen temeljac razvoja softvera. Kada su zahtjevi ispunjeni, krećemo na slijedeću fazu – dizajn, koji ima ulogu plana za implementaciju danih zahtjeva. Kada je dizajn završen, implementaciju realiziraju programeri. Napredujući prema kasnijim stadijima implementacijske faze, vršimo integraciju različitih komponenti softvera koje su proizvedene od strane različitih timova. Kada su završene implementacijska i integracijska faza, softverski proizvod se testira i debugira. Ovdje se uklanjaju sve greške koje su nastale u ranijim fazama. Nakon toga se softverski proizvod instalira i kasnije održava.

Kod Waterfall modela ne smijemo preskakati faze. Na slijedeću fazu možemo krenuti tek kada je prethodna kompletno završena i ispravna.

2.1.3 Prednosti Waterfall modela

Bolje utrošeno vrijeme u ranijim fazama kod ovog modela može voditi do veće ekonomičnosti kasnije. Npr. ako se u ranijim fazama otkrije greška, saniranje je mnogo ekonomičnije nego ako se ista greška otkrije tek u kasnijim fazama, čime štedimo novac, vrijeme i trud. Slijedeći pozitivni argument Waterfall modela je stavljanje naglaska na dokumentaciju i izvorni kod softverskog proizvoda. Isto tako neki preferiraju Waterfall model zbog njegovog jednostavnog i discipliniranog pristupa. Ovaj model napreduje linearno kroz diskretne, lako razumljive i lako objašnjive faze, te je zato jako jednostavan. Isto tako pruža lako obilježive kontrolne točke u razvojnom procesu.

2.1.4 Nedostatci Waterfall modela

Mnogi smatraju da je Waterfall model loša ideja u praksi, uglavnom jer je uvriježeno mišljenje da je nemoguće usavršiti jednu fazu u razvoju softverskog proizvoda, pa tek onda krenuti na drugu. Većina softverskih projekata moraju biti otvoreni promjenama zbog eksternih faktora; većina softvera je napisana kao dio ugovora sa klijentom, a klijenti su skloni promjenama svojih ranijih zahtjeva. Zbog toga je veliki problem mijenjati proizvod u kasnijim fazama.

Također je teško procijeniti vrijeme i troškove za svaku fazu. Isto tako Waterfall model ne pokriva nikakve formalne mjere kontrole menadžmenta nad projektom.

Još jedan važan nedostatak koji treba uočiti je loša iskorištenost resursa. Samo određen broj članova tima će biti kvalificiran za svaku fazu; neki članovi koji su specijalizirani za jednu fazu, ne rade ništa za vrijeme drugih faza.

2.2 Ekstremno programiranje

Ekstremno programiranje je softverska inženjerska metodologija, najpoznatija od nekoliko brzih razvojnih softverskih metodologija. Ova metoda omogućuje prilagođavanje promjenama zahtjeva u bilo kojem trenutku u procesu razvoja projekta, te je time realističnija i ima bolji pristup nego metoda koja definira sve zahtjeve na početku projekta, čime se smanjuju troškovi realizacije.

XP propisuje day-to-day praksu za menadžere i razvojne programere; ta praksa je namijenjena da utjelovljuje i potiče softver više razine koji nastaje u interakciji klijenta i razvojnog tima, te tako dobivamo bolju kvalitetu usluge jer u potpunosti ostvarujemo razumijevanje s klijentom.

2.2.1 Ciljevi XP-a

XP je definiran kao:

- Pokušaj da uskladi ljudski faktor i produktivnost
- Mehanizam za socijalne promjene
- Put prema poboljšanju
- Stil razvoja
- Softversku razvojnu disciplinu

Glavni cilj XP-a je smanjenje troškova promjene. U tradicionalnom sistemu metode razvoja zahtjeva za projekt su određene na početku razvoja projekta i često fiksne od te točke pa na dalje. To znači da će trošak promjene zahtjeva u kasnijoj fazi biti visok. XP smanjuje troškove promjene uvođenjem osnovnih vrijednosti, principa i prakse. Prihvaćanjem XP-a, sustav razvoja projekta trebao bi biti fleksibilniji s obzirom na promjene.

2.3 Dizajn uz korisničko iskustvo (User experience design)

Ovaj dizajn je podskup na polju iskustvenog dizajna, koji se odnosi na kreiranje arhitekture i modela interakcije, koji imaju utjecaj na korisničku percepciju uređaja ili sustava. Opseg polja ove metode je usmjeren prema svim aspektima korisničke interakcije s proizvodom. Korisničko iskustvo je izraz koji se koristi za opisivanje svih iskustava i zadovoljstava korisnika kada koriste neki proizvod ili sustav. Obično se to odnosi na kombinaciju softvera i poslovnih tema, npr. prodaja preko weba.

Dizajn s korisničkim iskustvom je integriran u razvoj softvera i ostale oblike razvoja aplikacija tako da se postigne interakcija s korisnikom, te napravi plan rada sukladno s korisnikovim ciljevima.

Glavne dobiti ove metode su:

- Smanjenje nepotrebnih opcija koje nisu potrebne korisniku
- Unapređenje opće iskoristivosti sustava
- Realizacija dizajna i razvoj projekta kroz detaljne i pravilno definirane smjernice

Glavni nedostatak ove metode je mogući nastanak problema kod determinacije klijentovih želja. Ako softverski inženjer u potpunosti ne shvati zadatak naručenog softvera ili ako dođe do nerazumijevanja između inženjera i klijenta, postoji mogućnost da se razvoj počne odvijati u krivom smjeru, što će zasigurno povećati trošak i vrijeme razvoja softvera.

3. Zaključak

U okviru ovog seminara dan je kratki pregled standarda, razvojnih faza i metoda koje se koriste kod razvoja softvera. Uglavnom sve analizirane metode sadrže slične faze pri razvoju. Glavna razlika ovih metoda je u pristupu prema realizaciji određenog problema. Kao što se pokazalo, različiti pristupi imaju svoje dobre i loše strane. Kod Waterfall modela razvoj teče kontinuirano iz faze u fazu, što rezultira otklanjanjem grešaka u ranim fazama razvoja, ali i slabijom mogućnosti procjene troškova i potrebnog vremena za razvoj softvera. XP metoda ima manju osjetljivost na promjene zahtijeva tijekom same realizacije projekta, što uvelike može smanjiti troškove, ali ima veću osjetljivost na *bugove* jer se faze ne realiziraju tako temeljito kao kod npr. Waterfall modela. Metoda koja se oslanja na korisničko iskustvo najveće napore ulaže u ostvarivanje kvalitetne komunikacije s korisnikom, te prepoznavanje onoga što korisnik želi. Pokazalo se u praksi da ovaj pristup nije uvijek pogodan zbog toga što su klijenti vrlo često „zbunjeni“ i ne mogu precizno i točno definirati namjenu koju bi njihov softver trebao imati.

Sve opisane metode su još uvijek u upotrebi, iako neke češće, neke rjeđe. Na programskom inženjeru ostaje da prema raspoloživim resursima i danim zahtjevima odabere odgovarajuću metodu kojom će se poslužiti pri razvoju softvera, kako bi se proces razvoja maksimalno optimizirao.

4. Literatura

- Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", 1986
- McConnell, Steve (2006). *Software Estimation: Demystifying the Black Art*. Microsoft Press.
- McConnell, Steve (2004). *Code Complete, 2nd edition*. Microsoft Press. McConnell, Steve (1996). *Rapid Development: Taming Wild Software Schedules*. Microsoft Press.
- Parnas, David, A rational design process and how to fake it ; An influential paper which criticises the idea that software production can occur in perfectly discrete phases.
- Kent Beck: *Extreme Programming Explained: Embrace Change*, Addison-Wesley
- Kent Beck and Martin Fowler: *Planning Extreme Programming*, Addison-Wesley
- Martin Fowler: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley