Lekcija 4 - Zahtevi

Karakteristike poslova oko zahteva

- Najteže je otkriti **šta tačno implementirati**!
- Ima najveći uticaj na ceo projekat u oba slučaja: i ako se uradi dobro i ako se uradi loše.
- Najteže ga je ispraviti u kasnijim fazama projekta.
- Po običaju, najbolje ga je raditi u iterativnom maniru.

Jedna naučna studija je pokazala da je većina softverskih projekata imala uzroke svojih grešaka u pogrešnim zahtevima: 56% grešaka zbog zahteva (greška u zahtevu ima kumulativan uticaj), 27% zbog dizajna, 7% zbog konstrukcije i 10% otpada na ostale uzroke. Cena ispravljanja greške se može eksponencijalno uvećati u zavisnosti kada/gde se otkrije.

Pojam zahteva se može tretirati i kao vrsta ograničenja, tj. kao *odluka* s kojom želimo kontrolisati način realizacije projekta. Da nema zahteva svaki slučajno generisan program bi bio validan, a ujedno i totalno beskoristan!

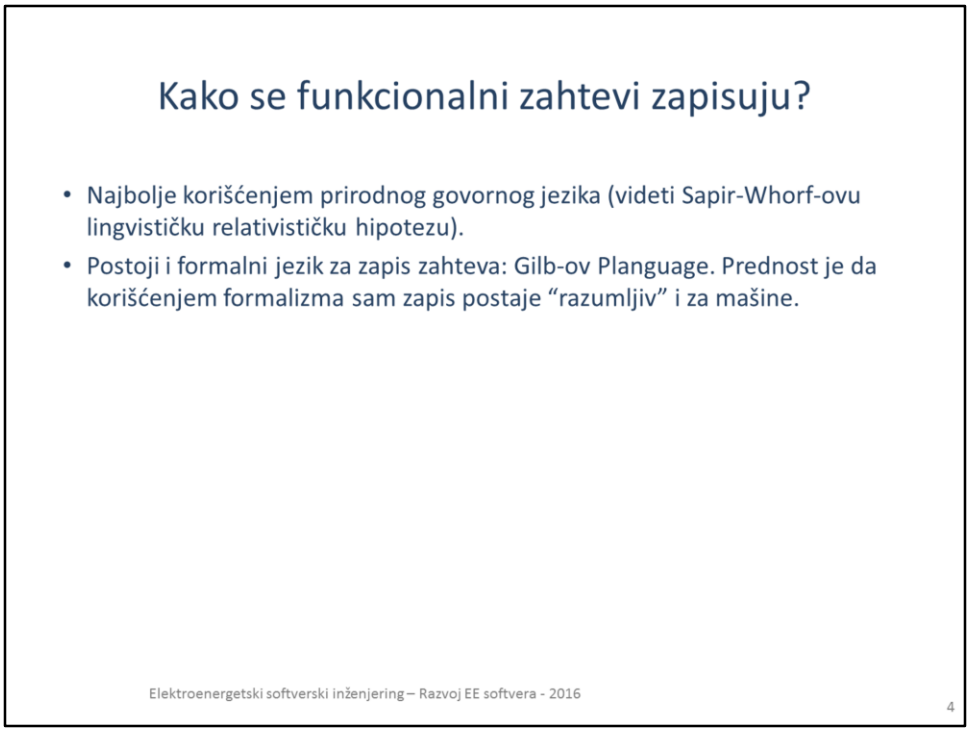
Tipovi specifikacija zahteva

- Funkcionalni zahtevi (**šta** sistem treba da radi sa aspekta klijenta) se zapisuju u funkcionalnoj specifikaciji sistema.
- Tehnički zahtevi se zapisuju u tehničkoj specifikaciji sistema.
- Nefunkcionalni zahtevi se zapisuju takođe u funkcionalnoj specifikaciji sistema.

Agilne metode koriste korisničke priče (user stories) za zapis zahteva.

Brza laboratorijska vežba: definišite zahteve za kuvalo! Ovo treba da pokaže kako se TDD princip treba primeniti i na zahteve, i koliko je besmisleno razdvajati funkcionalne od nefunkcionalnih zahteva, odnosno koliko je ime ovog drugog nakaradno.

Osnovna poruka je međutim ista: pre pisanja ijedne linije koda treba definisati zahteve!



Korišćeni jezik ne određuje samo šta govorite, veći i šta možete da kažete (odnosno mislite). Na primer, kod nekih domorodačkih zajednica u Africi ne postoji izraz za brojeve veće od tri. Svi brojevi veći od tri su *mnogo*. U takvom jeziku teško da se može razviti numerička analiza.

Inače, ova pojava se direktno može prevesti i na izražajnu moć programskih jezika. Zbog toga u knjizi Code Complete, 2nd Edition, autor knjige razlikuje "programming in a language" i "programming into a language" (za diskusiju videti http://stackoverflow.com/questions/8279605/meaning-of-program-into-your-language-and-program-in-your-language). Da biste definitivno proširili svoje vidike (izražajnu moć) treba da naučite neki od funkcionalnih programskih jezika. Dobra knjiga za to je *Structure and Interpretation of Computer Programs* (https://mitpress.mit.edu/sicp/).

Struktura funkcionalne specifikacije

- Rezime (overview) – kratak opis šta proizvod radi
- Napomena (disclaimer) - uglavnom govori o statusu specifikacije. U početku treba da naglasi da je ona u fazi izrade (draft).
- Ime autora (jedna osoba) – rukovodilac razvoja (tech lead) ili projekt menadžer
- Nekoliko tipičnih primera korišćenja proizvoda – pospešuje interakciju sa klijentom (ovde su UML sekvencijalni dijagrami veoma korisni)
- Prikaz izgleda korisničkih ekrana (mockup screens) i njihovih veza – veoma korisno za razvojni tim i ljude koji rade na izradi tehničke dokumentacije
- Spisak stvari šta proizvod **ne** uključuje (out of scope)
- Otvorene stavke – tokom vremena se ona smanjuje
- Ideje oko dizajna i osobina proizvoda – sve što sporedno padne na pamet tokom izrade specifikacije zahteva, a tiče se implementacije ili marketinga
- Buduće stavke (backlog) - sve što će se razmatrati u narednoj verziji

Postoje alati za izradu modela ekrana (wire frames), kao što je MockupScreens (mockupscreen.com). Za laku izradu sekvencijalnih dijagrama autor uglavnom koristi SDEdit (edit.sourceforge.net), jer se dijagrami definišu u intuitivnom tekstualnom formatu.

Funkcionalna specifikacija je u većini slučajeva 100% potpuna i tačna na kraju same implementacije. Zato se treba uvek potruditi naći balans između dovoljno detaljne specifikacije za početak implementacije i sigurnosti da je sve bitno obuhvaćeno i shvaćeno propisno.

Tipovi zahteva u funkcionalnoj specifikaciji

- Korisnički – osobine softvera, prikazi interakcija sa sistemom, itd.
- Domenski – pravila domena (uglavnom za regulisana tržišta)
- Nefunkcionalni (non-functional)
- Nezahtevi (non-requirements)

Korisnici veoma teško znaju da izraze tačno šta žele. To je opšti problem i kod ekstrakcija znanja eksperata. Oni često ne znaju da artikulišu kako su došli do rešenja (to ima veze za R režimom rada našeg mozga). Daleko je češće da će korisnici reagovati na stvari koje ne žele, odnosno žele drugačije. Zato je pojam rudarenja (digging) zahteva daleko prikladnije od prikupljanja (elicitation) zahteva (ne može se tek tako pitati korisnik “Šta i kako želite da program radi?”).

Suština je otkriti prave poslovne potrebe korisnika. Često se to može postići analizom zašto je nešto bitno za klijenta i zašto ga radi onako ga radi. Nije dovoljno samo reprodukovati trenutno ponašanje. Možda klijent radi sada stvari onako kako može, a ne zato što tako želi!

Veoma je opasno implementacijske odluke (kako?) pretvoriti u zahteve.

Zašto je prikupljanje (rudarenje) zahteva teško?

- Problem granice sistema (scope)
- Problem razumevanja – dihotomija domena klijenta i softverskih inženjera
  - sistem inženjer u sredini kao prevodilac
  - klijent kao deo razvojnog tima (omiljena kod Agilnih metoda)
- Problem promenljivosti zahteva – zato je potrebno kontrolisati izmene zahteva (Change Control Board)
- Problemi koji nisu tehničke prirode – uglavnom politički problemi između raznih departmana klijenta, ili unutar razvojne organizacije.

Analiza zahteva

- Klasifikacija (grupisanje) po oblastima
- Rangiranje po prioritetu
- Ispitivanje u odnosu na druge zahteve:
  - konzistentnost sa ciljevima projekta
  - važnost (da li je zaista bitno isporučiti ga u ovoj verziji)
  - testabilnost
  - izvodljivost sa aspekta ciljne platforme, performanse, i svih ostalih tzv. nefunkcionalnih zahteva
  - jasnoća (veoma usko vezana za testabilnost)