Načrtovanje in razvoj informacijskih sistemov

Specifikacija zahtev za programsko opremo

TEORIJA IN PRAKSA

IKTPROJEKT.SI

Specifikacija zahtev za programsko opremo

TEORIJA IN PRAKSA

UREDIL: **Mitja Kovačič, spec**.

mitja.kovacic@iktprojekt.si

Maribor, 2020

slike: www.pexels.com

Kazalo vsebine

„Good requirements are a myth“

UVOD

Dokumentiranje, specifikacija zahtev za programsko opremo je prva, zelo pomembna, aktivnost pri razvoju programske opreme. Ob poznavanju agilnega manifesta ter opazovanju dela mladih razvojnih skupin, samo dokumentiranje programskih zahtev danes nima več tako močan vpliv na uspeh načrtovanja in razvoja programske opreme. Kljub temu lahko takoj povem, da to početje zagovarjam še danes in upam, da podobno mislečim lahko predstavim praktične smernice za to delo.

Brošura prinaša teorijo in izbrane praktične smernice za prenos ideje o delovanju programske opreme (aplikacije) od naročnika, produktnega vodje ali sistemskega analitika k načrtovalcem in razvijalcem programske opreme in informacijskih rešitev.

Še pred tem bralcem predstavim korake razvoja informacijskih sistemov in nekaj načinov, kako jih lahko izvajamo. Brošuro zaključujem s opisom pomena testiranja zahtev in s tem zagotavljanjem kakovosti programske opreme.

Kdo narekuje zahteve?

Narekovanje zahtev poteka na različnih nivojih in tudi okoljih (*vir: ISO/IEC/IEEE 29148).*

Najprej je tu ***zunanje okolje***. To predstavljajo razni trendi in konkurenca na trgu, predpisi in zakonodaja, tehnološki in varnostni standardi ter kulturne značilnosti okolja.

Organizacijsko okolje predstavljajo poslovne strategije in načrti, politike in interni standardi ter domenske tehnologije in organizacijska kultura.

Poslovni procesi in operacije (omejitve in pravila) predstavljajo ***poslovno okolje***. To obsega tudi standarde kakovosti izvajanja poslovnih funkcij.

Sistemskemu okolju lahko rečemo kar informacijski sistem. Ta v večini primerov obsega nabor medsebojno povezanih računalniških aplikacij, programsko opremo.

Na najnižjem nivoju se srečamo z ***aplikativnim okoljem*** in implementacijo same specifikacije zahtev za programsko opremo.

Upravljanje zahtev

Skrbi zagotavljanje ažurnosti zahtev lahko rečemo tudi ***upravljanje zahtev***. To lahko v praksi poteka v 5-ih korakih:

1. Odkrivanje

   Prepoznamo različna okolja in deležnike ter informacijske vire. Odkrijemo zahteve za informacijsko rešitev.

2. Analiza

   Prepoznamo predpostavke, omejitve, odvisnosti, zunanje vmesnike in povezave ter pomembnost zahtev.

3. Dokumentiranje

   Izdelamo prototipe in tekstualni opis zahtev.

4. Validacija

   Pregledamo kakovost opisa zahtev, skladnost z lastnostmi dobre specifikacije in odpravimo morebitne dvoumnosti. Potrdimo zahteve.

5. Vodenje sprememb

   Pregledamo in sprejmemo zahteve ter ažuriramo opis.

Ali je agilno brez opisa?

V zaključku poglavja nadaljujem z uvodno mislijo. Pred časom sem slišal naslednje: *»Sedaj bomo delali agilno in specifikacija zahtev več ne bo potrebna.«*

To čisto ne bo držalo. V tej brošuri so podane smernice tudi za tak način razvoja programske opreme.

Tisti, ki ste zavezani javnim naročilom in naročate programsko opremo, pa brez specifikacije zahtev za programsko opremo zagotovo ne bo šlo.

RAZVOJ INFORMACIJSKIH SISTEMOV

Različni avtorji delijo razvojni cikel informacijskih sistemov (rešitev, aplikacij) na več korakov, ki jih tudi različno poimenujejo. Sam povzemam te korake in jih nekako poimenujem na podlagi praktičnih izkušenj. Izpostavljene so ključne naloge (ne vse), ki so v posameznih korakih v neposredni povezavi s *Specifikacijo zahtev za programsko opremo (SZPO)*.

1. Sistemska analiza

   Sistemsko analizo opisujem v ločenem poglavju.

   - zbiranje informacij in spoznavanje problema
   - prepoznava in definiranje sistemskih zahtev
   - izgradnja prototipa in dokumentiranje zahtev
   - določitev prioritete zahtev
   - kreiranje in ovrednotenje alternativ

2. Oblikovanje sistema

   Načrtujemo in ustrezno oblikujemo arhitekturo sistema ter njegovo sklopljenost z drugimi rešitvami, sistemi.

- oblikovanje integracije v omrežju
- oblikovanje sistemske arhitekture
- oblikovanje uporabniških vmesnikov
- oblikovanje sistemskih vmesnikov
- oblikovanje in definiranje podatkovne zbirke
- izdelava prototipa za predstavitev detajlov
- oblikovanje in integracija sistemskih kontrol

3. Razvoj sistema

   Načrtujemo in razvijemo (programiramo) module sistema in njihovo integracijo, skladno s standardi kodiranja.

   - načrtovanje programskih modulov
   - izdelava funkcijskih specifikacij
   - programiranje (kodiranje) programskih modulov
   - statično testiranje (metoda bele škatle) modulov in integracije

4. Testiranje sistema

   Skrb za kakovost poteka v vseh korakih. Testiranje opisujem v ločenem poglavju.

   - načrtovanje testiranja
   - dinamično testiranje (metoda črne škatle) sistema
   - verifikacija lastnosti (funkcionalnosti) – „Check List“

5. Stabilizacija sistema

   Odprava morebitnih neskladij na podlagi uporabniške izkušnje (končni uporabniki, administratorji, lastnik).

   - načrtovanje namestitvenih procedur
   - prenosi in konverzije podatkov
   - test sprejemljivosti (UAT)
   - priprava in izobraževanje uporabnikov in skrbnikov

Opisani koraki predstavljajo klasični razvojni cikel informacijskih sistemov, znan kot tudi SDLC – System (Software) Development Life Cycle.

Zasledili boste tudi korake kot so načrtovanje razvoja (obsega, ekipe), uvajanje in šolanje, vzdrževanje… Sam sem se pri definiranju korakov osredotočil na samo umestitev specifikacije zahtev v razvojni cikel.

Zapis ne govori o fazah informacijskega (IT) projekta, pri katerem je programska oprema eden izmed objektnih ciljev (izdelkov, dobav).

Več na temo načrtovanja in organiziranja ter vodenja informacijskih projektov vam lahko predstavim ob našem srečanju na kavi.

V nadaljevanju opisujem tri načine (pristope) izvajanja SDLC korakov, rečemo lahko tudi faz.

- Prvi je zaporedno izvajanje korakov.
- Drugi je izvajanje korakov v več (tudi 10) krajših ponovitvah (iteracijah, ciklih, sprintih).
- Tretji pa je izvajanje korakov v treh, nekoliko daljših ponovitvah, ki temelji na prototipnem pristopu.

Kako izvajati SDLC?

Izbira pristopa izvajanja SDLC je odvisna od samega IT projekta.

Če postavimo na tehnico predvidljiv (preroški, Waterfall) pristop ali prilagodljiv (adaptivni, Agile) pristop, dobimo naslednji rezultat (izbor):

- Predvidljiv SDLC

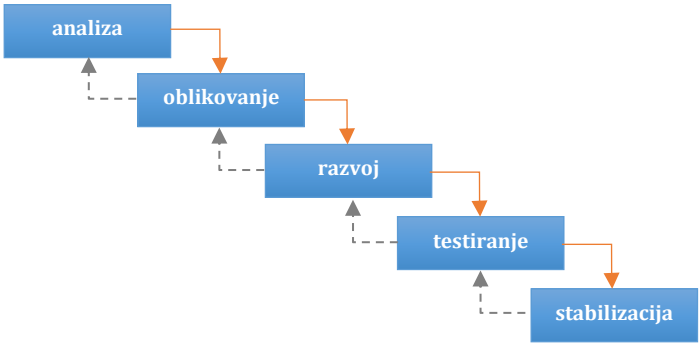
  Zahteve so dobro razumljene in nedvoumno definirane, opisane, obstajajo nizka tehnična tveganja.

- Adaptivni SDLC

  Nedoločene zahteve in potrebe po rešitvi, obstajajo visoka tehnična tveganja.

Predvidljiv (slapovni, Waterfall) pristop

Koraki razvoja si zaporedno sledijo eden drugemu. So medsebojno odvisni in zaključek prejšnjega koraka avtorizira naslednji korak.



Slika prikazuje opisan model. Vsak zaključen korak prinese tudi povratno informacijo uspešnosti predhodnega koraka.

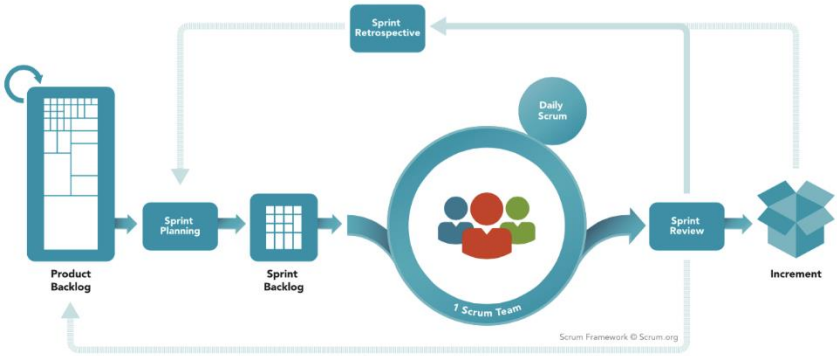
Tak pristop ne predvideva ponovitev in ni primeren za obsežen projekt. Uporabijo ga tisti, ki se lotevajo razvoja aplikacij na tradicionalni (predvidljiv) način. Imajo čas in so natančno analizirali naročnikov posel, potrebe in problem. Poznajo (skoraj) vse njegove zahteve in vedo (beri "želijo"), da se ne bodo spreminjale.

Adaptivni (Agile) pristop

Glavni moto agilnih metodologij je, da je poudarek na razvoju produkta, delujoči programski kodi in zavedanju, da se uporabniška izkušnja s časom spreminja. Leta 2001 so neodvisni strokovnjaki, ki so se ukvarjali z različnimi agilnimi metodologijami, napisali Manifest agilnosti, ki pravi (*vir: scrum.org*):

- Posamezniki in interakcije nad procesi in orodji
- Delujoča programska oprema nad celovito dokumentacijo
- Sodelovanje s stranko nad pogajanji o pogodbenih določilih
- Odziv na spremembe nad upoštevanjem načrtov

Koraki razvoja si sledijo v okviru dogovorjenega cikla (iteracije, tipično 2 do 4 tedne), ki jih imenujemo "Sprint". Na začetku ekipa določi količino dela, ki ga lahko opravi v enem sprintu. Naloge določi glede na vrednosti iz prioritetnega seznama, ki ga imenujemo "Product Backlog". Izbrane naloge prenese v seznam nalog posameznega cikla oz. "Sprint Backlog". S kratkimi dnevnimi sestanki "Daily Scrum", preverja napredek in odpravi morebitne ovire.



Slika prikazuje opisan model po SCRUM (*vir: scrum.org*)

Razvojna skupina mora biti zelo homogena in oblikovana tako, da doseže največjo produktivnost. Skupina se sama organizira in je sama odgovorna, da opravi vse naloge, ki jih je načrtovala za posamezni Sprint.

Pri tem pristopu vemo, da naročnik v času razvoja ne bo podal vseh zahtev. Tudi pisanje jim preveč ne diši. V več iteracijah razvoja aplikacije bodo zapisali uporabniške zgodbe (v enem stavku), ki bodo sproti narekovale seznam lastnosti aplikacije. Prilagajali se bomo naročnikovim željam.

Prototipni pristop

Prototip v fazi razvoja informacijske rešitve predstavlja izdelek, verzijo programske opreme, ki je namenjena za testiranje in validacijo posameznega gradnika ali celote ter za verifikacijo izdelave končne verzije izdelka.

Kaj prinaša pristop?

Ocenjujem, da lahko s takim pristopom razvoja programske opreme zaključim z delom v treh ponovitvah (dveh verzijah prototipov in tretje, končne verzije).

Pri razvoju upoštevam same lastnosti predvidljivega pristopa in prednosti adaptivnega. Navajam nekatere ugotovitve.

- Vztrajanje pri sistemski analizi

  Kljub odporu ali nerazpoložljivosti deležnikov vztrajam pri učinkovito izvedeni sistemski analizi. Seveda je velika verjetnost da vseh zahtev ne bom odkril, zajel, vendar poskušam omejiti ta korak na začetek v največjem možnem obsegu in še na dve izvajanji v nadaljevanju v zelo omejenem obsegu.

- Omejitev razvoja verzije

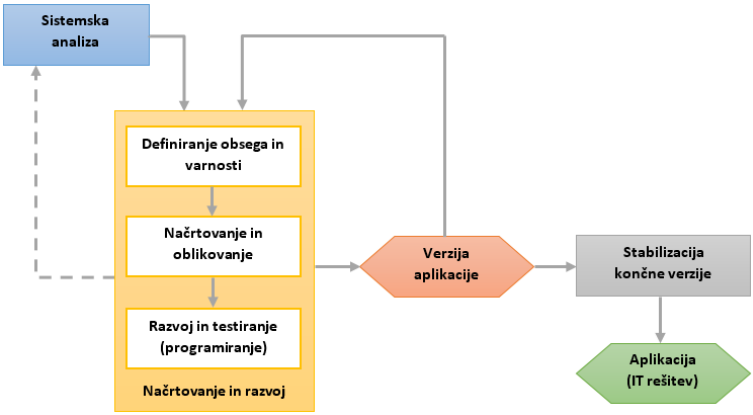
  Trudim se jasno definirati trajanje in obseg razvoja prve in obeh ostalih verzij. Znotraj tega koraki razvoja informacijske rešitve potekajo bolj ali manj zaporedno.

- Verzija aplikacije

  Pride čas, ko prenesem odgovornost na uporabnika. Zahtevam temeljito testiranje, preverjanje skladnosti razvitih funkcionalnosti z dogovorjeno specifikacijo zahtev za verzijo. Na tem mestu se tudi konča ponovitveni cikel.

- Stabilizacija končne verzije

  Zadnji korak predvideva odziv na izvedeno funkcionalno testiranje in na uporabniški test sprejemljivosti programske opreme (User Acceptance Test).



Slika prikazuje lastni pristop, ki temelji na prototipnem pristopu razvoja aplikacij.

SISTEMSKA ANALIZA

Sistemska analiza (SA) - ključna faza razvoja informacijskih sistemov, ki v osnovi prinaša odkrivanje in analizo ter dokumentiranje zahtev za programsko opremo. Izdelek SA je dokument ***Specifikacija zahtev za programsko opremo (SZPO)***, ki predstavlja popoln opis zunanjega obnašanja programske opreme.

S tehniko iskanja dejstev prepoznamo (konkretiziramo) uporabniške potrebe in te zapišemo v obliki specifikacije zahtev, ki jih podkrepimo z raznimi diagramskimi tehnikami.

Kdo je sistemski analitik?	Nekdo, ki je odgovoren za odkrivanje, analiziranje in dokumentiranje zahtev ter komunikacijo med deležniki.

Izvaja sistemsko analizo in vodi aktivnosti skozi proces upravljanja z zahtevami.

Odkrivanje in analiza zahtev

»Zbiranje« zahtev nakazuje nek urejen način prepoznavanja zahtev, čeprav so te največkrat skrite. To počnemo zelo redko.

Bolj točen opis pridobivanja zahtev bi lahko bil ***»izvlačevanje«*** zahtev. Oba pojma sta lahko zavajajoča, zato nalogo zajemanja zahtev imenujem kar **odkrivanje zahtev**.

Tehnike odkrivanja zahtev, ki jih tudi sam uporabljam so:

1. Intervju

   Zaprta (z natančno definirani odgovori) in odprta (mišljenja in ideje) vprašanja.

2. Vprašalnik

   Intervju z različnimi ciljnimi skupinami za pridobitev nabora mišljen o zahtevah iz vidika skupine.

3. Delavnice

   Skupinsko odkrivanje primerov uporabe, zahtev in demonstracija njihove preverljivosti.

4. Pregled dokumentacije

   Uporabo informacij in dokumentacije obstoječih sistemov, kot so uporabniška navodila, ekranske slike ali obrazci.

5. Opazovanje

   Prepoznavanje neučinkovitosti na obstoječih delovnih nalogah, ki temeljijo na ročnem ali informacijsko podprtem izvajanju.

Začnem z vprašalnikom, ki ga prilagodim ugotovitvam na predstavitvenem intervjuju. Priprava takega vprašalnika zahteva dokaj velik vložek, ki se pa povrne v ponovni uporabi vprašalnika.

	Opis komponent	Izvajanje storitev	SLA
Fizični strežniki			
Diskovni sistem			
Omrežje			
Virtualizacija			

Na sliki je predstavljen izsek vprašalnika za primer uporabe obstoječih tehnologij za zagotavljanje izvajalnega okolja (podatkovni center).

Prototipi v sistemski analizi

Prototipiranje v sistemski analizi je orodje, ki nam olajša prepoznavanje in razumevanje zahtev za programsko opremo. Za prototip uporabimo ti. statični (opisuje strukturo, zgradbo) ali dinamični (opisuje obnašanje, funkcije) diagram.

Kaj je prototip v sistem. analizi?

Diagram. V nadaljevanju izpostavljam nekaj diagramov, ki jih največkrat uporabimo za dokumentiranje zahtev.

Procesni diagram

Predstavlja grafični prikaz toka poslovnega procesa (funkcije, storitve), ki ga želimo informatizirati.

Osnovni gradniki, simboli diagrama

1. Aktivnost

   Predstavlja korak (nalogo) v procesu.

2. Usmerjene povezave

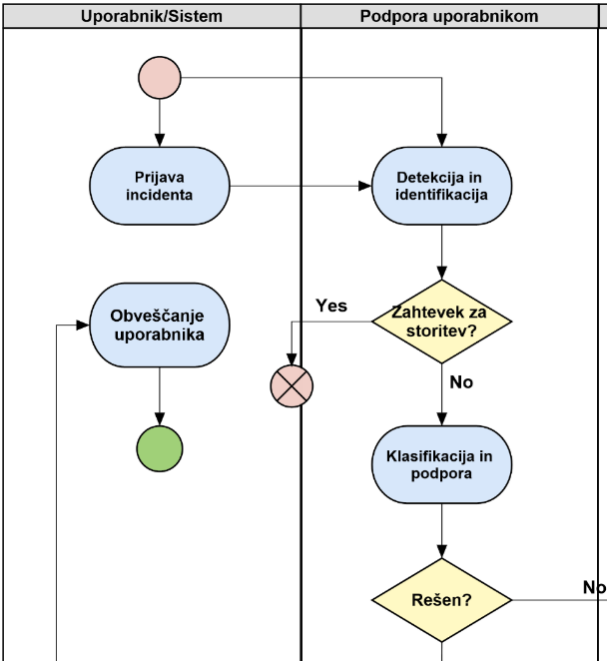
   Predstavljajo procesni tok in povezave med aktivnostmi.

3. Pogojne vejitve

   Odločitve, ki temeljijo na vprašanju. Procesni tok se deli glede na odgovor na vprašanje (največkrat »da« ali »ne«).

4. Zunanji viri

   Predstavlja zunanje vire, kot so dokumenti, datoteke, vhodi in izhodi.



Na sliki je predstavljen izsek procesnega diagrama za primer vodenja incidentov.

Blokovni diagram

Predstavlja grafični prikaz strukture sistema, ki je sestavljen iz več komponent (modulov, nivojev) oziroma drugih manjših sistemov.

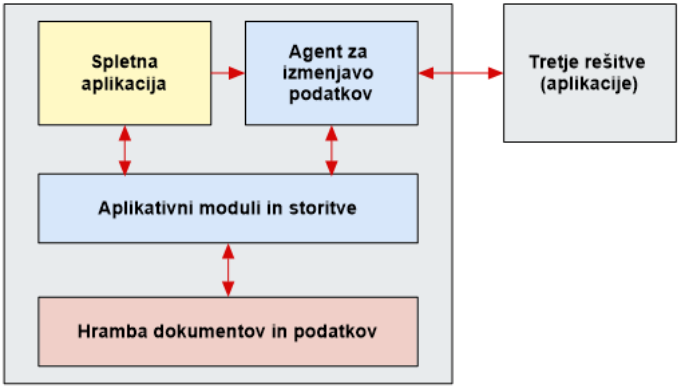
Gradniki diagrama

1. Bloki

   Definira gradnik (modul, podsistem) sistema.

2. Usmerjene povezave

   Predstavljajo povezave in tok med bloki.



Na sliki je predstavljen blokovni diagram za dokumentno usmerjeno aplikacijo (računovodska aplikacija).

Diagram primerov uporabe

Procesni in blokovni diagram, sta pomagala pri razumevanju poslovnega in sistemskega okolja. Za razumevanje aplikacijske (programske) domene pa priporočam diagram primerov uporabe. Gre za zajemanje in grafično predstavitev zahtev iz vidika uporabnika.

Osnovni gradniki diagrama so (*vir: FERI Maribor*)

1. Igralec (akter)

   Definira vlogo posameznika ali drugega sistema, ki je v interakciji s sistemom, ki ga opisujemo.

2. Primer uporabe

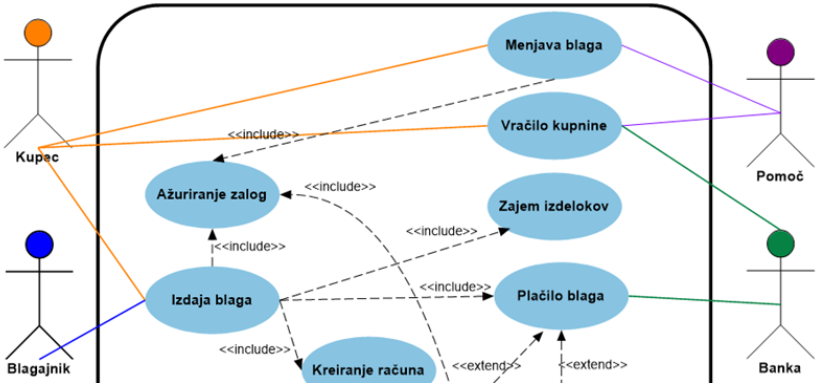
   Predstavlja zaporedje transakcij v sistemu in opisuje možen potek interakcije med sistemom in enim ali več akterjem.

3. Povezave med akterji in primeri uporabe

   Asociacija, ki pomeni udeleženost oziroma komunikacijo med akterjem in primerom uporabe.

4. Sistem (ali podsistem)

   Ograjuje opisane primere uporabe, lahko tudi aplikacija,



Na sliki je predstavljen izsek diagrama primerov uporabe za primer, sistem trgovinskega poslovanja (prodaje blaga).

Zelo pomembno je, da ločujemo pojmovanje - prototip (diagram) v fazi sistemske analize, ki ga opisuje to poglavje in prototip (produkt) v fazi razvoja programske opreme, ki je opisan v prejšnjem poglavju.

Ekranske slike (tudi Mockup)

Še vedno se držim pravila, da slika (diagram) lahko pove več kot *»tisoč besed«*.

Za izdelavo ekranskih slik uporabljam dva pristopa

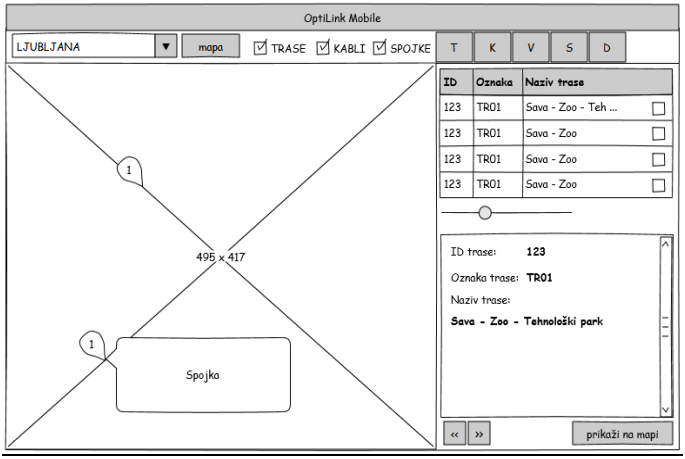
1. Skiciranje

   Izdelava ekranskih slik s pomočjo Mockup orodij.

2. HTML stran

   Izdelava ekranskih slik z uporabo HTML5, CSS3 in Javascript razvojnega ogrodja (npr. Bootstrap)

V prvem primeru ekranske slike pripravim hitreje. Lahko tudi simuliram izvajanje aplikacije. V drugem primeru potrebujem več časa. Tak način pa prinaša prednosti pri razvoju (implementaciji), saj že vključuje fazo načrtovanja in grafičnega oblikovanja aplikacije.



Na sliki je predstavljena ekranska slika (Mockup) za nadzorno ploščo tablične aplikacije.

Uvod v dokumentiranje zahtev

Zahteva predstavlja nek pogoj, zmožnost, ki ga vaša aplikacija mora izpolnjevati, da bo dosegla svoj namen. Te velikokrat ni enostavno prepoznati in jih razumeti. Še težje jih je zapisati (dokumentirati) na nek enostaven in razumljiv način.

Kaj je dobra zahteva?

Dobra zahteva mora predstavljati nedvoumno opisano funkcionalnost programske opreme, ki uporabnika podpira pri njegovem delu.

Dokument SZPO vključuje natančen opis obsega programske opreme ter jasne in razumljive detajle za načrtovanje in razvoj programske opreme. Je enotno mesto za ažuren opis dejstev za razvoj programske opreme in prinaša povezavo med zahtevami uporabnikov in funkcionalnosti.

Koristi dokumentiranja zahtev

1. Razumljena povratna informacija.

   Specifikacija zahtev je dokument, ki naročniku jamči, da ponudnik načrtovanja in razvoja programske opreme razume problematiko, ki jo želi naročnik rešiti s to rešitvijo. Zato je dokument zapisan v naravnem jeziku, brez dvoumnosti in na obema razumljiv način.

2. Razčlenitev problema na obvladljive dele.

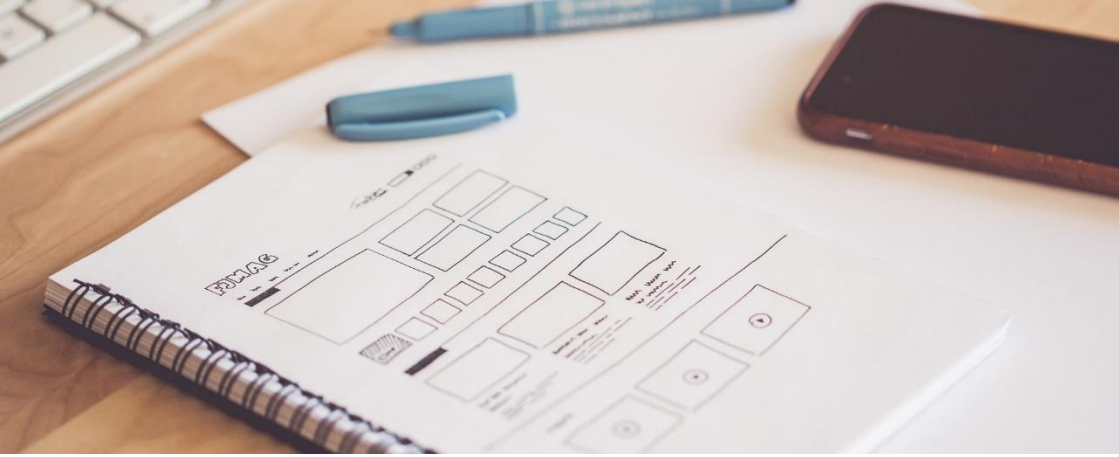
   Dobra praksa organiziranosti specifikacije zahtev (standard) sama po sebi člani opis programske opreme na manjše dele. Ti so opisani na strukturiran način in jih naročnik tudi lažje obvladuje.

3 Zahteve za oblikovanje.

Specifikacija zahtev vsebuje opisane detajle, ki vplivajo na načrtovanje, oblikovanje in razvoj sistema. Ti opis naročniku pomagajo, da se pravilno načrtuje arhitektura, potrebna infrastruktura ter gradniki (moduli, storitve) programske opreme z uporabniškimi vmesniki in njihova sklopljenost.

4 Krovni dokument naročila (razpisa).

Specifikacija zahtev prinaša opis zunanjega obnašanja programske opreme, ki naročniku pomaga pri definiranju pogodbenih obveznosti s ponudnikom in načrtovanju in izvedbi zaključnega – prevzemnega testiranja produkta.



PRISTOPI DOKUMENTIRANJA ZAHTEV

Dokument *Specifikacija zahtev za programsko opremo - SZPO,* prinaša popoln opis zunanjega obnašanja programske opreme. Odgovarja na vprašanje »KAJ« želimo imeti in ne »KAKO« deluje programska oprema.

Gre za najobsežnejše poglavje, ki prinaša opis lastnosti in vsebinskih gradnikov samega dokumenta s primeri. Opisan je tudi postopek dokumentiranja zahtev v obliki uporabniških zgodb ter postopek dokumentiranja varnostnih zahtev iz vidika informacijske varnosti.

Lastnosti dokumenta

V nadaljevanju opisujem nekaj ključnih lastnosti dokumenta SZPO s priporočili in primeri. Specifikacija naj bo:

1. popolna in natančna
2. konsistentna
3. merljiva
4. preverljiva
5. določljiva s prioritetami

Popolna in natančna

Zahteve ne smejo biti dvoumno ali slabo definirane ampak zapisane popolno in natančno. Naj opisujejo uporabniške potrebe v skladu z poslovnimi potrebami in so še vedno razumljive vsem deležnikom.

> *„Aplikacija mora biti integrirana s plačnim sistemom.“*

Zahteva je zapisana preveč splošno.

S katerim plačilnim sistemom? Ali ima plačilni sistem na razpolago API vmesnik?

To sta samo dve vprašanji, ki se porajata pri taki zahtevi. Kako bi zapisali na nedvoumen način (ena sama interpretacija zahteve):

> „Aplikacija uporablja PayPal kot plačni sistem.“

Primer različnih interpretacij zaradi postavitve ločil:

> *„Aplikacija računa hitrost avtomobila in njegovo pot, v manj kot 5s.“*
> *„Aplikacija računa hitrost avtomobila, in njegovo pot v manj kot 5s.“*

Primer razrešitve nedoločnosti (TBD):

> *„Višina dodatka v odstotkih bo znana po sprejetju zakona (zakaj ne poznamo odgovora).“*
>
> Kdo je odgovoren za razrešitev in kdaj?
>
> *„Višina dodatka v odstotkih bo znana po sprejetju zakona. Zakon sprejme DZ, predvidoma decembra 2020 in začne veljati 1.1.2021.“*

Primer uporabe formalnega jezika (formule):

> *„Znesek z DDV = Znesek brez DDV * (1 + Stopnja DDV/100)“*

Primer enostavnega zapisa zahtev:

> *„Uporabnik se prijavi v trgovino in pregleda svoja trenutna naročila.“*

Dve zahtevi ali več.

> *„Uporabnik se prijavi v trgovino z uporabniškim imenom in geslom.“*
>
> *„Uporabnik pregleda seznam svojih trenutnih naročil.“*

Priporočilo:

- izdelaj pojmovni slovar (natančno določimo pomen izrazov)
- upoštevaj pasti naravnega jezika (ločila)
- uporabljaj formalni jezik za specifikacijo zahtev
- izogibaj se ITD, ITN …
- v primeru nedoločnosti uporabi TBD (To Be Defined)
- piši enostavne, kratke in jedrnate stavke
- modalni glagoli niso uporabni (lahko, mora, naj …)
- uporabljajmo aktivne in ne pasivne oblike

Konsistentna

Zahteve ne smejo biti niti podvojene niti protislovne. V primeru konflikta mora biti izbira, navodilo, tako da so vse zahteve jasne vsem deležnikom.

Primer neskladni zahtevi:

> *„Vnos podatkov sklopa B je na voljo izključno takrat, ko je tudi na voljo vnos podatkov sklopa A.“*
>
> *„Vnos podatkov sklopa A se začne takoj, ko se zaključi vnos podatkov sklopa B.“*

Primer podvojeni zahtevi.

> *„Najvišji dvig gotovine na bankomatu je 1.000 €.“*
>
> *„Znesek dviga gotovine na bankomatu je med 100 in 1.000 €.“*

Priporočilo:

- izogni se neskladnosti opisa posebnih lastnosti objektov (npr. izpisi, forme, reporti, obrazci …)
- izogni se logičnemi neskladnosti med dvema opisanima akcijama (npr. zaporedje akcij)

Merljiva

Zahteve morajo biti opisane na osnovnem nivoju funkcionalnosti (osnovni elementarni proces) iz vidika uporabnika. Zahteve višjega nivoja poskušamo pretvoriti na nižji nivo. Vsaka vključena zahteva se naj da preveriti (testiranje, meritve). Nedvoumne zahteve niso preverljive.

Primer elementarnega procesa:

> *„Poslovni proces zahteva dobavo funkcionalnosti upravljanja s podatki zaposlenih.*

Taka zahteva se razdeli na manjše enote dela (poišči, poglej, dodaj, ažuriraj, briši zaposlenega…).

Primer nemerljive zahteve:

> *„Seznam zaposlenih (poročilo) se generira in izpiše v doglednem in uporabniku sprejemljivem času.“*

Primer merljive zahteve:

> *„Seznam zaposlenih (poročilo) se generira in izpiše v 20 sekundah v 60% primerih. V ostalih primerih se generira in izpiše v 30 sekundah.“*

Priporočilo:

- pri odkrivanju zahtev izhajaj iz nivoja elementarnih (osnovnih) procesov – uporabniških aktivnosti

Preverljiva

Zahteve so preverljive, če velja, da se vsaka vključena zahteva lahko preveri (testiranje, meritve). Nedvoumne zahteve niso preverljive.

Primeri nepreverljivih zahtev:

„Sistem mora biti zmogljiv.“

„Aplikacija mora pokazati dobre performanse.“

„Aplikacijski vmesnik je uporabniku prijazen.“

Primeri preverljivih zahtev:

„Stran se naloži v 5-tih sekundah.“

„Aplikacija naloži stran v 5-tih sekundah.“

„Izborni podatki – šifranti so prikazani v spustnih seznamih.“

„Vnos datuma poteka z izbiro iz prikazanega koledarja.“

„Vsak vnosni element ima na voljo pripadajočo pomoč.“

Priporočilo:

- ne uporabljaj pojmov „lep“, „dober“, „odličen“, „prijazen“, „učinkovit“ …
- uporabljaj merljive in kvantitativne pojme

Določljiva s prioritetami

Vsaka posamezna zahteva mora imeti identifikator, ki določa njeno pomembnost. Vse zahteve za programsko opremo niso enako pomembne. Nekatere so bistvene (tudi kritične) za programsko opremo, druge se lahko obravnavajo kot želje uporabnikov.

Priporočilo:

- Obvezne: UAT ne bo potrjen
- Pogojne: izboljšujejo produkt, implementirane kasneje, UAT bo potrjen
- Opcijske: možnost ponudniku, da predlaga kaj več

User Acceptance Test (UAT) je zaključni test sprejemljivosti programske opreme iz vidika uporabnika.

Druge lastnosti dokumenta SZPO so še unikatno določljiv, stabilen, sledljiv in skladen (opis najdemo v nadaljevanju navedenih standardih).

Struktura dokumenta

Ključna so štiri poglavja dokumenta SZPO. Vsa poglavja obsegajo podpoglavja, gradnike dokumenta (*vir: IEEE 830-1998)*.

1. Uvod

   Pregled (overview) dokumenta.

   1. *Namen*
      Opišemo namen dokumenta in določimo ciljni avditorij.
   2. *Obseg*
      Poimenujemo programski produkt, ki naj bi ga zgradili. Razložimo tudi, kaj bo programski produkt delal in (po potrebi) česa ne bo.
   3. *Kratice, definicije, akronimi*
      Podaja definicije vseh izrazov, akronimov in okrajšav, potrebnih za pravilno interpretiranje specifikacije.

4. *Sklicevanja*
   Priskrbimo za popoln seznam dokumentov, na katere se sklicujemo v specifikaciji ali v drugih ločenih, vendar v specifikaciji navedenih, dokumentih,

5. *Pregled vsebine*
   Opišemo nadaljnjo vsebino dokumenta in razložimo njegovo organiziranost.

2. Splošen opis

Opis splošnih (generalnih) dejavnikov, ki vplivajo na produkt in njegove zahteve. Ne vsebuje podrobnega opisa zahtev ampak predstavlja neko ozadje za te zahteve, ki omogoča njihovo lažjo razumevanje.

1. *Perspektive produkta*
   Podamo povezave programske opreme z drugimi sorodnimi produkti ali projekti. Opišemo ali je produkt neodvisen in popolnoma samostojen ali del večjega sistema.

2. *Funkcije produkta*
   Vsebuje povzetek funkcij, ki jih bo programska oprema zagotavljala.

3. *Značilnosti uporabnikov*
   Opišemo tiste značilnosti potencialnih uporabnikov, ki bodo vplivale na oblikovanje specifičnih zahtev.

4. *Omejitve*
   Opis vsebuje splošen opis vseh postavk, ki bodo razvijalca omejevale pri načrtovanju sistema (regulativna politika, strojne omejitve, vmesniki do drugih aplikacij, revizijske funkcije, varnost …).

5. *Domneve in odvisnosti*
   Vsebuje seznam vseh dejavnikov, ki vplivajo na zahteve, podane v specifikaciji. Ti dejavniki niso omejitve pri

načrtovanju, temveč so pomembni zato, ker njihova sprememba zahteva spremembo dokumenta.

6. *Pomembnost zahtev*
   Zapišemo zahteve, katerih implementacija je pogoj za sprejemljivost produkta, katere se implementirajo naknadno in katere se lahko odložijo za naslednjo verzijo programskega produkta.

3. Specifične zahteve

Detajlni opis vseh zahtev produkta, ki načrtovalcem omogoča oblikovanje (načrtovanje) produkta in testerjem testiranje zahtev. Vsaka zahteva je ločeno zaznana od uporabnikov, administratorjev in sistema (ali povezanih sistemov).

1. *Funkcionalne zahteve*
   Specificiramo, kako naj bodo vhodi v programski produkt transformirani v izhode. Opisujemo temeljne akcije, ki morajo biti zajete v programski opremi. Organiziranost poglavja (funkcionalne zahteve) predstavlja razčlenitev programske opreme na obvladljive dele, za katere opišemo zahteve delovanja, obnašanja.

   Primeri strukturiranja opisov:

   - Procesni vidik (po poslovnih procesih)
   - Uporabniški vidik (po vrstah uporabnikov)
   - Funkcijski vidik (po primerih uporabe)
   - Modularni vidik (po hierarhiji modulov)

   Pri tem si pomagamo z izbranim pristopom prototipiranja v sistemski analizi.

2. *Zahteve za zunanje vmesnike*
   Programska oprema z okoljem komunicira prek zunanjih vmesnikov. Te delimo v 4 skupine in sicer: uporabniški

vmesniki, vmesniki strojne opreme, vmesniki programske opreme in komunikacijski vmesniki.

3. *Zahteve za zmogljivost*
   Numerično podamo statične in dinamične zahteve za programsko opremo ali za uporabniško interakcijo s programom kot celoto.

4. *Logične zahteve za podatkovni model*
   Opišemo zahteve za podatkovno bazo, ki mora biti razvita kot del produkta. Opišemo tudi običajne in posebne operacije, ki jih zahteva uporabnik.

5. *Omejitve pri načrtovanju*
   Opišemo omejitve pri načrtovanju, ki so »vsiljene« prek drugih standardov, omejitev glede strojne opreme itd.

6. *Značilnosti IT okolja*
   Največkrat opišemo nefunkcionalne zahteve ali lastnosti aplikativnega okolja (infrastrukture, platforme) kot so npr. zanesljivost, razpoložljivost, varnost, možnost vzdrževanja, prenosljivost.

4. Priloge

   Prilog ne obravnavamo vedno kot dela specifikacije zahtev in zato niso nujno potrebne. Vsebujejo lahko dodatne skice, diagrame, izpise, obrazce, poročila in druge dokumente. Predstavljajo pomožne informacije za bralce.

Struktura dokumenta je povzeta po IEEE 830-1995.

Kako zapisati zahteve?

V uvodu sem predstavil možnost organiziranja zahtev po nivojih (različne vrste zahtev).

Poslovne zahteve so zahteve najvišjega nivoja, ki so povzete iz poslovnega procesa informatizacije – projekta.

Primer (na kratko):

„Informacijski sistem za zagotavljanje storitev mobilnega bančništva banke. Uporabniki, ki so komitenti banke izvajajo transakcije prenosa denarja in plačevanja računov. Ostali uporabniki izvajajo transakcije prenosa denarja.

- *Banka: Informacijski sistem za zagotavljanje storitev mobilnega bančništva*
- *Komitenti: Uporabniški račun, prenos denarja in plačevanje računov*
- *Ostali: Uporabniški račun in prenos denarja“*

Uporabniške zahteve (primeri uporabe), so zahteve iz vidika uporabnika (dateljne, nižji nivo) in določajo arhitekturo oblikovanje sistema.

Primer:

„Uporabniki, ki so komitenti banke izvajajo plačevanje računov.

- *Uporabnik ima na nadzorni plošči na voljo seznam prejetih računov registriranih podjetij. Uporabnik pogleda račun, plača račun, zavrne račun ali ga umakne iz seznama.*
- *Plačevanje računov izvaja uporabnik ali njegov skrbnik.“*

Sistemske zahteve predstavljajo najnižji nivo opisa zahtev. Gre za natančni opis vsake od posameznih zahtev. Lahko se uporabi oblika uporabniških zgodb.

Primer:

„Registracija podjetij, ki posredujejo račun v spletno banko (elementarni proces). Zahteve modula:

- *naziv podjetja (dobavitelja),*
- *povezava z uporabnikom (identifikacijska številka)*

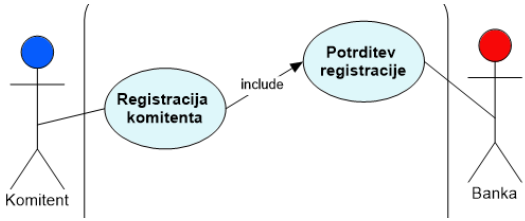
- *avtomatsko plačilo – Da/Ne*
- *plačilo v celoti – Da/Ne*
- *omejitev zneska plačila – ne plača se račun z višjim zneskom“*

Kako opišemo zahteve?

Pomagamo si z diagrami iz sistemske analize ter kratko in jedrnato opišemo aktivnosti, akcije iz diagrama.

Imamo poslovno zahtevo in diagram primerov uporabe:

„Informacijski sistem za zagotavljanje storitev mobilnega bančništva. Vsi zainteresirani uporabniki, komitenti za mobilno bančništvo se morajo predhodno registrirati. Po potrjeni registraciji postanejo uporabniki mobilnega bančništva.“



Slika prikazuje diagram primerov uporabe. Na podlagi diagrama lahko strukturiramo poglavje, ki opisuje specifične zahteve.

V skladu z opisano strukturo lahko dobimo naslednje (izvleček iz dokumenta):

3.1. Funkcionalne zahteve mobilnega bančništva

Funkcionalne (specifične) zahteve za aplikacijo, ki zagotavlja informacijsko podporo storitvam mobilnega bančništva.

3.1.1. Register komitentov

Vodenje zbirke registriranih uporabnikov, komitentov za uporabo aplikacije mobilno bančništvo.

3.1.1.1. Registracija uporabnika (komitenta)

Identifikacija uporabnika, komitenta in evidentiranje zahtevka za uporabo aplikacije mobilno bančništvo.

FS01: Spletna stran in registracija

(ne opisujem zahteve)

FS02: Obrazec (forma) za registracijo

Obrazec omogoča vnos podatkov v treh korakih. V prvem koraku se vnesejo osebni podatki bodočega uporabnika. V drugem koraku se potrdi razumevanje splošnih pogojev in obdelave osebnih podatkov. V tretjem koraku se preverijo vneseni podatki in hrani zapis.

Vnos podatkov novega, bodočega uporabnika poteka v skladu s tabelo (izvleček):

Podatek	Opis	Format	Obv?
Elektronski naslov	Elektronski naslov, unikatno uporabniško ime	Niz E-mail oblike zapisa	DA
Geslo	Geslo uporabnika	Niz od 6 do 15 znakov in vsaj ena velika črka ter številka	DA
Izpisano ime	Izpisano ime/naziv uporabnika	Niz do 30 znakov	NE

3.1.1.2. Potrditev registracije

Registracije uporabnikov pregleda uredniški odbor. Ta preveri (je/ni komitent) in potrdi ali zavrne registracijo. Vsi zainteresirani s potrjeno registracijo postanejo uporabniki mobilnega bančništva.

Kaj lahko gre narobe?

Predstavil sem lastnosti dobre SZPO. Te lahko tudi predstavljajo pasti pri opisovanju zahtev.

Tabela v nadaljevanju predstavlja primer slabo (pomanjkljivo) in dobro zapisane zahteve glede na posamezno lastnost.

Slaba zahteva	Dobra zahteva
Ni natančno definiran pojem (dvoumnost)	
Registracija uporabnika … Kreiranje računa … Zainteresirani …	Razlaga pojmov (definicije), poglavje 1.3. ▪ Zainteresirani: vsak … ▪ Uporabnik: vsak, registriran in potrjen …
Ni konsistentna (vrstni red)	
Registracija se zaključi v vsakem koraku, zaželeno po zadnjem. Registracija se lahko zaključi po zadnjem koraku vnosa podatkov.	Vnos podatkov v obrazec registracije se zaključi izključno po zaključku tretjega koraka.
Ni preverljiva	
Prijazen obrazec za registracijo s pomočjo.	Vsak vnosni element ima na voljo pripadajočo pomoč, ki je dostopna prek ikone (?).
Ni merljiva	
Registracija se naj preveri v uporabniku sprejemljivem času.	Vneseni podatki registracije se preverijo v največ 5 sekundah.

Kaj pa je funkcijska specifikacija?

Dokument *Funkcijska specifikacija* prinaša popoln opis notranjega obnašanja programske opreme. Odgovarja na vprašanje »KAKO« naj deluje programska oprema. Lahko predstavlja natančen opis posameznih funkcijskih sklopov iz poglavja 3.1. in služi kot navodilo

programerjem (koderjem), skrbnikom podatkovnih zbirk, sistemskim skrbnikom (infrastruktura). Dokument strukturirano opisuje in vsebuje:

1. poslovni ali sistemski subjekt
2. primere uporabe (diagrami)
3. ekranske slike (mockup)
4. podatkovno relacijski diagram
5. informacijske poglede (izpise)
6. testni načrt

Zahteve kot uporabniške zgodbe

Omenil sem že, da je eden izmed razlogov za agilni pristop razvoja programske opreme hitra dobava produkta. Ta ni možna, če se srečamo s pomanjkanjem in nejasnostjo zahtev. Te pa lahko predstavimo z zgodbami.

Lastnik produkta zagotovi seznam zgodb, ki jih lahko pišejo vsi člani razvojne skupine. Te beležimo kjer koli in jih lahko združujemo na tabli (board) v programski opremi za agilno vodenje projektov.

Kaj je zgodba?

Zgodba (Story) je kratek in enostaven opis funkcionalnosti programske opreme, izražen z vidika uporabnika, skrbnika ali sistema.

Zapišejo se v obliki:

Kot <tip uporabnika>, hočem <nek cilj>, da <nek razlog>.

Primer uporabniške zgodbe.

„Kot komitent, moram kreirati svoj uporabniški račun, z namenom identifikacije in registracije za uporabo mobilnega bančništva.“

Uporabniške zgodbe združujemo v **epe** (Epic) in ti so združeni v okviru posamezne **teme** (Theme). Na drugi strani razvojna skupina posamezno zgodbo razdeli na **opravila** (Task).

Slika prikazuje primer organiziranosti dokumentiranja zgodb in njihovo delitev na opravila (*vir: www.knowledgehut.com*).

Sam način zapisa zgodbe določa njihovo prioriteto. Zgodbe se po prioritetah razvrstijo v skupine in obvezne so najprej realizirane, razvite. Primer poimenovanja, razvrščanja v skupine:

- obvezne (must)
- potrebne (should)
- zaželene (could)
- nepotrebne v verziji (won't)

Nasveti za pisanje dobrih zgodb

Uporabniške zgodbe so najpopularnejša agilna tehnika opisa zahtev za programsko opremo. Te pa ni enostavno napisati. Pri pisanju vam lahko pomagajo povzeti nasveti (*vir: www.romanpichler.com*).

1. Uporabniki na prvem mestu

   Zgodbe se vedno pišejo iz vidika uporabnika. Če ne vemo, kdo je uporabnik, ne moremo pisati uporabniških zgodb.

2. Poosebljanje za odkrivanje zgodb

   Kdo so uporabniki? Spoznajmo karakteristike uporabnikov na podlagi poosebljanja. Opišemo, kdo je oseba (uporabniki) in zakaj oseba (uporabnik) želi uporabljati programsko opremo.

3. Skupinsko kreiranje zgodb

   Z zgodbami nikoli ne upravlja izključno razvojna ekipa. Nastajajo skupaj z lastnikom produkta.

4. Enostavne zgodbe s pomenom

   Po predstavljeni predlogi zapišemo enostavne in razumljive zgodbe.

5. Začnimo s epi

   Najprej predstavimo grob prikaz obsega produkta.

6. Izpilite zgodbe to potankosti

   Ukvarjajmo se z zgodbo, dokler ni pripravljena (nedvoumna, izvedljiva z možnostjo demonstracije – testa).

7. Dodajte kriterije sprejemljivosti

   Kriteriji obogatijo zgodbo, ji omogočijo testiranje in demonstracijo uporabnikom in drugim deležnikom. Uporabite 3 do 5 kriterijev.

8. Uporabljajte kartice

   Uporablja se tudi izraz kartice zgodb (uporabniške zgodbe so predstavljene na kartici). Tehnika ima več prednosti, so poceni in se enostavno uporabljajo, lajšajo skupinsko delo in enostavno jih združujemo v skupine (tabla, zid, ekran) ter prikazujemo odvisnosti.

9. Vidne in dosegljive zgodbe

   Ne skrivajte zgodb na skupnih virih (diskih), na internetu ali v nepregledni in nedostopni programski opremi.

Uporabniške zgodbe niso vse. So zato, da omogočajo hiter razvoj programske opreme.

---

10. Ne zanesite se izključno na uporabniške zgodbe

    Zgodbe pomagajo k opisu funkcionalnosti programske opreme. Prikazujejo uporabniški vidik, niso pa dovolj za kreiranje uporabniške izkušnje. Iz vidika arhitekture programske opreme (sodelovanja komponent ali storitev) je potrebno dodati tudi sistemske (tehnične) zgodbe.

Za hitro izdelavo prototipa ali ekranske slike zgodbe še niso potrebne. To lahko naredimo brez njih. Na drugi strani tudi ne predstavljajo sam dokument Specifikacija zahtev za programsko opremo, so le njegov gradnik.

Zahteve informacijske varnosti

Eden izmed učinkovitih pristopov zmanjšanja varnostnih tveganj informacijskih sistemov je uvedba formalnega postopka (procesa) načrtovanja in razvoja, ki vključujejo dobre prakse informacijske varnosti. V izogib ranljivosti informacijskih sistemov uporabimo varen (Secure) SDLC, ki vsebuje varnostne kontrole v treh korakih.

- Modeliranje groženj (Threat Modeling) v fazah odkrivanja zahtev in oblikovanja sistema;

- Statično varnostno testiranje aplikacije (Static Application Security Testing - SAST) v fazah implementacije in testiranja;
- Dinamično varnostno testiranje aplikacije (Dynamic Application Security Testing - DAST) v fazah testiranja in uvedbe;

Samo testiranje predstavljam v zadnjem poglavju. Kot rečeno, pa do varnostnih zahtev pridemo s tehniko modeliranja groženj.

Modeliranje groženj

Kaj so grožnje in ranljivosti?

Grožnja pomeni možnost uspešne izrabe ranljivosti programske opreme (aplikacije) in ustvarja tveganje.

Ranljivost je šibka točka programske opreme (aplikacije), ki se lahko namerno ali nenamerno izkoristi. Vodi v tveganje.

Modeliranje groženj je proces prepoznavanja in dokumentiranja groženj računalniškim aplikacijam, katere zmanjšujemo v procesu implementacije.

- Na začetku procesa se seznanimo z željenimi funkcionalnostmi aplikacije skozi pregled faz odkrivanja zahtev in oblikovanje aplikacije.
- V odvisnosti od namena aplikacije, vrste shranjenih podatkov, posredovanja in procesiranja podatkov v aplikaciji in motivov napadov razvijemo scenarije in primere uporabe in tudi zlorabe, ki nam lahko pokažejo možnost napada na aplikacijo.
- Tako dobljen model groženj pregledamo in po potrebi posodobimo zahteve funkcionalnosti in oblikovanja aplikacije s ciljem zmanjšanja groženj aplikaciji.

Izvleček iz modela groženj prinaša varnostne cilje, seznam groženj in ranljivosti ter varnostne zahteve.

Varnostni cilji (in omejitve) morajo upoštevati načela informacijske varnosti (zaupnost, celovitost, razpoložljivost). Predstavljajo tudi del nabora projektnih ciljev. Pri določanju ciljev je potrebno razumeti namen in cilje potencialnih napadalcev.
U uporabo detajlov koraka prepoznavamo posamezne grožnje za scenarije in vsebino (vzrok za incident, ustvarjajo tveganja).

Primeri ciljev:

„Zaščititi in varovati osebne in poslovne podatke pred napadalci.“

„Doseči zahtevan nivo zagotavljanja storitve (SLA).“

„Zaščititi in varovati kredibilnost e-poslovanja podjetja.“

Seznam groženj in ranljivosti aplikacije – prepoznane in opisane so grožnje in ranljivosti aplikacije.

Primer strukture seznama:

1. kategorija grožnje – ljudje, narava, okolje
2. vir in naziv (tudi opis) grožnje – heker, zaposleni, potres, vročina
3. motivacija in aktivnosti
4. ukrepi nižanja (preventive)
5. ranljivost

Varnostne zahteve aplikacije – prepoznane in opisane so varnostne zahteve za načrtovanje in razvoj aplikacije.

Primeri varnostnih zahtev:

„Uporabnik se identificira s predhodno registriranim in potrjenim uporabniškim imenom, ki je elektronski naslov uporabnika.“

„Uporabniško geslo se pred hrambo v podatkovno zbirko pretvori v SHA1 hash obliko.“

„Vsak izpis (poročilo), ki vsebuje osebne podatke mora vsebovati enolični identifikator s pripadajočim zapisom v revizijski sledi o datumu, času in kreatorju izpisa.“

Varnost računalniških aplikacij pomaga odpraviti morebitne grožnje računalniškim aplikacijam.

Standardi za dokumentiranje zahtev

Postopek dokumentiranja (specificiranja) zahtev za programsko opremo si lahko poenostavite in uporabite dobro prakso. Te so predstavljene v različnih standardih. Na kratko predstavljam tri take standarde.

IEEE 830-1998

IEEE Recommended Practice for Software Requirements Specifications – opisuje priporočila pristopa k dokumentiranju (specifikaciji) zahtev za programsko opremo iz praktičnega vidika.

IEEE 1233-1998

IEEE Guide for Developing System Requirements Specifications – prinaša praktična navodila za proces razvoja in vodenja zahtev za programsko opremo.

ISO/IEC/IEEE 29148-2011

Systems and software engineering — Life cycle processes — Requirements engineering – opisuje proces in izdelke obvladovanja zahtev za programsko opremo skozi celote življenjski cikel informacijskih sistemov in programske opreme.

Ali bomo zmogli?

Ali ne bomo zamudili rokov, če se posvetimo specifikaciji?

Bomo, če

- trošimo čas na analizi in dokumentiranju zahtev za programsko opremo, ki jo nameravamo obdržati;
- največ 20% projektnega časa namenimo dokumentiranju zahtev;
- Obdržimo 5% projektnega časa ažuriranju dokumenta po začetku oblikovanja programske opreme.

In še 10 namigov za pisanje

Pisanje Specifikacije zahtev za programsko opremo zahteva čas!

1. Vzemite si čas za točen in nedvoumen opis.
2. Avtor naj ne bo programer (razvijalec) ampak nekdo, ki razume potrebe posla in jezik razvijalcev.
3. Avtor mora imeti dobre komunikacijske sposobnosti.
4. Vključite slike in diagrame za obrazložitev potreb.
5. Ažurirajte dokument takoj po spremembi zahtev ali njihovih prioritet.
6. Hranite na deljenih virih za lažji dostop celotni ekipi.
7. Uporabite za testiranje in validacijo.
8. Pišite za ciljne skupine.
9. Poskrbite, da vsebuje vse, kar potrebujejo načrtovalci in razvijalci.
10. Sklicujte se na povezane vsebine in dokumentacijo.

TESTIRANJE ZAHTEV

Testiranje zahtev poteka sproti, od njihovega nastanka pa vse do začetka uporabe posamezne funkcionalnosti programske opreme. Čim kasneje v razvojnem ciklu programske opreme odkrijemo napako, tem višji so stroški za njeno odpravo oz. več nas odprava napake stane.

Faza	Relativni stroški odprave napak
Specifikacija zahtev	0.1 – 0.2
Načrtovanje	0.5
Kodiranje	1
Testiranje modulov	2
Test sprejemljivosti	5
Vzdrževanje sistema	10

Tabela prikazuje relativne stroške odprave napačnih zahtev v posameznih razvojnih fazah in po uvedbi programske opreme.

Mnogo napak ostane neodkritih še precej časa po koncu faze, v kateri so bile narejene. Več kot polovica napak (54% delež napak) se odkrije šele po fazi kodiranja in testiranja enot (modulov) – pri

sistemskem in funkcionalnem testiranju. Večina napak (45% delež) izhajala kot posledica slabše opravljenih faz sistemske analize in oblikovanja sistema. Le manjši del napak (9% delež) je bilo povzročenih v fazi kodiranja s strani programerjev.

Napake, narejene pri specifikaciji zahtev, so po navadi nepravilna dejstva, izpuščena dejstva, nedoslednost in dvoumnost.

Tip napake	Delež v % glede na vse napake
Nepravilno dejstvo	49
Izpuščanje	31
Nedoslednost	13
Dvoumnost	5
Zahteva na napačnem mestu	2

Tabela prikazuje delež povzročenih napak glede na vrsto (tip) napake.

Večina napak je zaradi slabe sistemske analize in nerazumevanja konteksta nepravilno zapisano dejstvo (zahteva). Velikokrat se med samim razvojem in testiranjem funkcionalnosti zahteve tudi dodajajo, na novo odkrivajo. Te predstavljajo izpuščene zahteve v fazi odkrivanja in dokumentiranja zahtev.

Pristopi testiranja zahtev

Predstavljena sta dva pristopa k testiranju in samo razmišljanje o testiranju že v fazi dokumentiranja zahtev. Opisan je tudi način testiranja varnostnih zahtev.

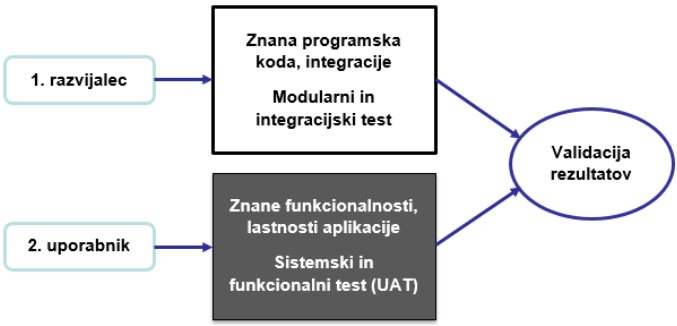
Kaj je testiranje?

1. Testiranje je aktivnost, ki pomaga najti hrošče, nepravilnosti in napake v programski opremi med razvojem le te z

namenom izdelave in zagotovitve varne in zanesljive rešitve za uporabnike.

2. Testiranje je proces, ki pomaga prepoznati pravilnost, kompletnost in kakovost (skladnost) razvite programske opreme.

Prvemu rečemo testiranje po metodi bele in drugemu testiranje po metodi črne škatle.



Slika prikazuje sinergijo obeh pristopa testiranja zahtev v fazi razvoja in delovanja programske opreme.

Testiranje po »beli škatli«	Testiranje po »črni škatli«
Tester ima dostop do razvojnega okolja, načrtovanja in razvoja aplikacije. Aplikacija je testirana od znotraj. Gre za pristop testiranja s strani razvojnika aplikacije.	Tester nima znanj s področja razvojnih tehnologij in razvoja aplikacij. Aplikacija je testirana od zunaj. Gre za pristop testiranja s strani uporabnika.
Zahtevana programska koda. Gre za analizo izvorne programske kode in modulov brez zaganja aplikacije.	Zahtevana delujoča aplikacija. Gre za analizo obnašanja zagnane aplikacije.
Najdene napake zgodaj v SDLC.	Najdene napake pozno v SDLC.

Nižji stroški odprave napak.	Višji stroški odprave napak.

Testiranje in specifikacija zahtev

Načrtovanje testiranja naj poteka po Specifikaciji zahtev za programsko opremo. Tam imamo zapisan primer uporabe programske opreme, ki lahko predstavlja testni scenarij.

Testni scenarij in primeri

Testni scenarij je največkrat dokaj nejasen zapis iz vidika testiranja, ki lahko obsega veliko število možnosti (primerov). Testiranje pa pomeni biti natančen!

Kako povezati SZPO in test?

Vsak testni scenarij ima svojo oznako. Uporabimo enako oznako, kot smo jo v specifikaciji zahtev.

Primer testnega scenarija in testnega primera:

FS02: Obrazec (forma) za registracijo

Vnos podatkov novega uporabnika:

- Preveri odziv v primeru vnosa veljavnega elektronskega naslova, gesla in imena
- Preveri odziv v primeru vnosa neveljavnega elektronskega naslova, gesla in imena
- Preveri odziv v primeru brez vnosa elektronskega naslova, gesla in imena

Drugi testni primeri v povezavi z zahtevo FS02 so še potrditev razumevanja splošnih pogojev ter preverjanje in hramba vnesenih podatkov.

Testni podatki

Samo prepoznavanje testnih podatkov je lahko dolgotrajen proces, saj moramo prepoznati podatke iz veljavnih in neveljavnih skupin pa tudi iz mejnih skupin (razredov). Včasih zahteva tudi kreiranje samih podatkov s strani testerjev.

Predpogoj in pričakovani rezultati

Za vsako uspešno testiranje morajo biti izpolnjeni predpogoji za izvedbo. Če taki pogoji niso izpolnjeni, testiranja v celoti ni možno izvesti, je pomanjkljivo in ne doseže svojega namena.

Primer predpogoja testiranja:

> Nameščena in delujoča mobilna aplikacija »ime aplikacije«. Vzpostavljen register (podatkovna baza) uporabnikov.

Navesti moramo tudi pričakovan rezultat testiranja. Če ne navedemo pričakovanega rezultata, se lahko zgodi, da zgrešimo napačne vrednosti, ki so na prvi pogled videti pravilne.

Primer pričakovanega rezultata testiranja:

> Preverjeni podatki na obrazcu ali opozorilo o nepravilnem vnosu ter dostop do objavljenih splošnih pogojev.

Koraki testiranja

Natančno opisan postopek testiranja s pripadajočimi vhodnimi podatki, rezultatom in verifikacijo. Same korake testiranja lahko sproti dopolnjujemo, tudi z zapisom komentarja.

Primer koraka testiranja:

Korak:	št. 2
Opis koraka:	obvezen vnos elektronskega naslova v polje »Elektronski naslov«

Podatki:	mitja.kovacic@iktprojekt.si, a@a.a, test@domena.si, (prazen niz) …
Rezultat koraka:	veljaven elektronski naslov
OK:	da
Komentar:	skladnost z vzorcem za e-naslov

Vrste testiranja programske opreme

V nadaljevanju je kratki opis raznih vrst testiranja pri razvoju in v času delovanja programske opreme. Specifikacija zahtev za programsko opremo je podlaga (vsebuje zahteve) za vse opisane teste.

Testi pri razvoju programske opreme so:

1. Modularni test (Unit Test)
   Nivo individualnega testiranja izvorne kode komponent (modulov) programske opreme.

2. Integracijski test (Integration Test)
   Nivo skupinskega testiranja sodelovanja, integracije programskih komponent (modulov).

3. Sistemski test (System Test)
   Nivo testiranja programske opreme v izvajalnem okolju v povezavi z ostalimi gradniki informacijskega sistema (strojna oprema, druga programska oprema, programski vmesniki itd).

4. Prevzemni test (Acceptance Test)
   Nivo testiranja programske opreme iz vidika uporabnikov z namenom ugotavljanja skladnosti z zahtevami na podlagi specifikacije zahtev.

Testi samega delovanja programske opreme pa so:

1. Funkcionalni test (Functional Test)
   Podobno kot prevzemni test, s katerim ugotavljamo skladnost programske opreme s specifikacijo zahtev. Tudi rečemo, da gre za QA (Quality Assurance) proces.

2. Performančni test (Performance Test)
   Testiranje odzivnosti programske opreme (izvajanje skript) v različnih odjemalskih okoljih, npr. spletnih brskalnikih.

3. Zmogljivostni test (Load Test)
   Testiranje odzivnosti programske opreme v realnih pogojih izvajanja, velikemu številu sočasnih uporabnikov. Poznamo Stress Test, ki nam pokaže odziv ob maksimalnem številu (obremenitvi) sočasnih uporabnikov.

4. Varnostni test (Security Test)

Testiranje varnostnih zahtev

O varnostnih zahtevah in njihovem odkrivanju sem pisal v prejšnjem poglavju. Informacijski varnosti programske opreme se največ posveča OWASP fundacija (www.owasp.org).

V okviru fundacije poteka projekt OWASP Top Ten, ki opisuje prvih 10 varnostnih tveganj spletnih aplikacij. Vsebuje tudi priporočila testiranja opisanih ranljivosti.

> Varnostni test (Security Test)
> Zelo pomembno testiranje z namenom odkrivanja ranljivosti programske opreme. Preprečevanje tveganj vdorov ter razkritja in poneverjanja podatkov (tudi brisanja).

Varnostni test izvajamo z etičnim vdiranjem v programsko opremo, informacijski sistem. Imenujemo ga tudi »Penetration Test«.

Kako lahko pomagamo?

Naš izziv in poslanstvo je pomoč pri organiziranju, načrtovanju in izvajanju informacijskih projektov.

1. Na podlagi vašega poslovnega primera izdelamo študijo izvedljivosti informatizacije poslovnih procesov skupaj s priporočilom za nadaljnje korake.

2. Izdelamo spletni prototip informacijske rešitve (aplikacije) in pripravimo specifikacijo zahtev za programsko opremo skupaj s pobudo za projekt.

3. Definiramo projektne cilje in pripravimo projektni načrt. Izdelamo zagonski elaborat ter pomagamo organizirati projektno ekipo in voditi projekt.

Kontakt: **www.iktprojekt.si**

Mitja Kovačič
IT Consultant, Project Manager,
System Architect, CSPM

Diplomiral sem na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, nato pa zaključil podiplomski specialistični študij na Fakulteti za organizacijske vede s področja organizacije in managementa informacijskih sistemov.

Od leta 1987 se ukvarjam z računalništvom in informatiko, v zadnjih dvajsetih letih pa z menedžmentom informacijskih sistemov in projektnim menedžmentom.

Leta sem opravljal tudi vodstvena dela v službi za informatiko. Vodim IT projekte informatizacije poslovnih procesov in elektronskega poslovanja ter projekte upravljanja s tveganji, uvajanja sistema informacijske varnosti ter upravljanja neprekinjenega poslovanja.

Sem tudi certificiran senior projektni menedžer in član slovenskega združenja za projektni menedžment ter predavatelj na višji šoli za informatiko.

https://www.linkedin.com/in/mitjakovacic/