SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

POSLIJEDIPLOMSKI DOKTORSKI STUDIJ

ELEKTROTEHNIKE I INFORMACIJSKIH TEHNOLOGIJA

KVALIFIKACIJSKI ISPIT

METRIKE I METODE MJERENJA I PROCJENJIVANJA U SOFTVERSKIM PROJEKTIMA

Mili Turić

Split, rujan 2022.

SADRŽAJ

1 UVOD

Danas se softveri nalaze svuda oko nas. U mnogim sferama svakodnevnog života, a da se počesto i ne primjećuju. Ljudi ih njima koriste se svjesno i nesvjesno, oslanjajući se na njih zdravo za gotovo i ne razmišljajući o njihovom životnom ciklusu. Softver je igrica koja se igra na mobitelu za vrijeme čekanja autobusa, kao i mobilna aplikacija pomoću koje se gleda vozni red tog autobusa, softver upravlja sustavima u automobilima, očitava sliku rendgenskog snimka, izvršava bankovne transakcije, predviđa vremensku prognozu, obrađuje slike s nadzornih kamera i još mnogo toga što je postalo normalno u životu. Vjeruje se da ti softveri rade ispravno, da su sigurni i pouzdani, skalabilni i dostupni. Da bi sve to i bilo (a često i nije) tako, potreban je izuzetan napor softverskih inženjera koji rade na njihovom razvoju.

Široka lepeza softvera koji su korišteni proteže se od jednostavnih do vrlo kompleksnih koji za sobom povlače i kompleksan razvojni put. Današnji softveri su integrirani s mnoštvom servisa i uređaja pa ih i to čini još kompleksnijima. Stoga je način izrade ali i održavanje takvog softvera kompleksan i zahtjevan. Posljedično je to onda i skupo pa je onda i očekivano da investitori kod inicijalizacije projekata razvoja softvera žele znati koliko će ih to stajati. Da bi se mogla dati takva informacija, potrebno je procjenjivati ključne elemente softvera koji će se razvijati pa na temelju tog doći do konačne cijene. Kada se to dodatno zakomplicira agilnim razvojem koji se jako malo bavi dokumentacijom, ponestaje izvora informacija nužnih za procjenu. Procjenjuju se funkcionalnosti i resursi u koje spadaju ljudi, vrijeme i novac, a kao spona svega tog na kraju će se vidjeti kvaliteta. Neku razinu kvalitete može se željeti, pa i procijeniti, ali će ju se u pravom smislu vidjeti po završetku razvoja ako ju se izmjeri. Slično tom je i sa svim ostalim parametrima koji se procjenjuju. Sve njih bi po završetku razvoja trebalo mjeriti i uspoređivati s procjenama kako bi se gradilo bazu takvih podataka i inženjersku disciplinu.

Stoga je važno rano procjenjivanje i korištenje metoda koje mogu dati pouzdanu procjenu na temelju oskudnih artefakata u što ranijoj fazi životnog ciklusa razvoja softvera.

U drugom poglavlju dan je pregled metodologija za razvoj softvera s naglaskom na agilni razvoj softvera koji je danas jako zastupljen. Opisuje se nekoliko značajnih metoda, a najznačajnija među njima je Extremno programiranje – XP. U trećem poglavlju predstavljene su metode mjerenja i opisan sam početak mjerenja korištenjem linija koda kao mjere funkcionalnosti (LOC), ali i neke druge metode nastale kasnije tokom razvoja ovog područja.

Naglašeni su problemi mjerenja i navedeni razlozi za procjenjivanje. U četvrtom poglavlju obrađene su metode procjenjivanja poput procjene po analogiji, procjene stručnjaka i druge. U petom poglavlju iznesen je problem i izazov procjenjivanja te istaknuta povezanost tehnika i algoritama strojnog učenja i umjetne inteligencije s procjenjivanjem. Zaključni osvrt dan je u šestom poglavlju.

2 METODOLOGIJE RAZVOJA SW

2.1 Metodologije općenito

Brojne su metodologije razvoja softvera koje su se pojavile od početka razvoja softverskog inženjerstva do danas ali se ipak sve ugrubo mogu svrstati u dvije skupine: klasične i agilne metodologije. Softverski timovi se organiziraju u skladu s nekom od metodologija, a metodologije slijede jedan ili više modela razvoja softvera. Ono što je karakteristično za današnji razvoj softvera je činjenica da se vrijeme od ideje do razvoja proizvoda, odnosno, do gotovog proizvoda sve više skraćuje. Proizvodi koji proizlaze iz ovih projekata često su jedinstveni i prilagođeni naručitelju pa gotovo niti jedan projekt nije isti kao prethodni, svaki ima neku svoju specifičnost. Stoga je za vođenje važan odabir metodologije koja će se koristiti [1], [2].

Klasične metodologije pojavile su se u samom početku razvoja softverskog inženjerstva, a one se dijele na funkcionalno orijentirane koje slijede logiku starijih funkcionalno orijentiranih programskih jezika (Fortran, Cobol, C) i objektno orijentirane metode koje slijede logiku objektno orijentiranih programskih jezika (C++, C#, Java) [3]. Svima su im zajednički pricipi vodopadnog modela koji je u stvari fikcija jer se u stvarnosti nikada baš tako ne događa. Gotovo uvijek imamo preklapanja između faza jer npr. aktivnosti poput dizajna krenu prije nego li je završen cijeli proces prikupljanja zahtjeva ili testiranje krene prije nego li je cijeli razvoj gotov [4]. S druge strane, agilne metodologije su se tek sredinom 1990-tih godina pozicionirale kao alternativa tradicionalnim metodologijama. Smatra se da su one pogodne za IT projekte jer su interaktivne i inkrementalne, te fleksibilne i s bržim odgovorima na promjene. Također, one su prilagodljive jer kada dođe do promjene tijekom njihove primjene, moguće je da konačni rezultat bude drugačiji od prvobitno zamišljenog [1]. Najpopularnija metodologija agilnog razvoja softvera je *Extreme programming* – XP [5], a pored nje poznate su još i *Scrum, Feature driven development* – FDD, *Dynamic systems development* – DSDM, *Crystal* itd. U ovom radu predstavljeno je nekoliko najznačajnijih metodologija.

Dvije spomenute skupine metodologija razlikuju se i po svojim svojstvima, a ovo su najznačajnije razike:

- Klasične metodologije razvoj softvera promatraju kao pažljivo planirani proces koji ima svoj početak i kraj, a u tom procesu su u potpunosti utvrđeni zahtjevi na temelju kojih

se određuje precizni plan aktivnosti, raspored ljudi i resursa uz uredno dokumentiranje svih aktivnosti i proizvedenih dijelova softvera.

- Agilne metodologije razvoj softvera promatraju kao kontinuirani niz iteracija kojima se produkt usklađuje s trenutnim zahtjevima koji se stalno mijenjaju jer ne postoji cjelovita specifikacija zahtjeva. Upravljanje projektom se ne provodi u pravom smislu riječi, nego se aktivnosti dogovaraju s korisnicima, a pri tom je planiranje kratkoročno i izbjegavaju se svi oblici dokumentacije jer se smatra da je program u svom izvornom obliku najpouzdanija dokumentacija [3], [6].

2.2 Agilni razvoj softvera

2.2.1 Principi agilnog razvoja

Temeljne vrijednosti agilnog razvoja razrađene su u 12 principa koji se mogu pratiti u razvoju softvera. Načela koja slijede temelje se na Agilnom manifestu [6], [7], [8]:

1. Zadovoljenje kupca ranom i kontinuiranom isporukom inkremenata,
2. Podjela velikih zadataka na manje zadatke koji se mogu brzo dovršiti,
3. Prihvaćanje činjenice da samoorganizirani timovi daju najbolje rezultate,
4. Motiviranje ljudi okolinom i podrškom koja im je potrebna,
5. Stvaranje procesa koji promiču održive napore,
6. Održavanje tempa za dovršetak radova,
7. Prihvaćanje zahtjeva za izmjenom, čak i u kasnim fazama projekta,
8. Kontinuirano komuniciranje projektnog tim i naručitelja tijekom cijelog projekta,
9. Poticanje atmosfere redovnog razmišljanja o tome kako biti učinkovitiji,
10. Mjerenje napretka na temelju završenog posla,
11. Kontinuirana potraga za izvrsnošću,
12. Korištenje promjena da bi se stekla konkurentska prednost.

2.2.2 Manifest agilnog razvoja

Sažetak manifesta je otkrivanje boljih načina za razvoj softvera radeći to i pomažući drugima da to čine. Kroz taj rad iskristalizirale su se četiri temeljne vrijednosti.

Četiri temeljne vrijednosti razvoja agilnog softvera, kako je navedeno u Agile manifestu, su [6], [7], [8]:

1. Pojedinci i interakcije idu za procesima i alatima,
2. Radni softver koristi opsežnu dokumentaciju,
3. Suradnja s kupcem važnija je od pregovora o ugovoru,
4. Odgovor na promjene važniji je od praćenja plana.

2.2.3 Extreme programming – XP

Extreme programing je metodologija koja je nastala 2000. godine, a autorstvo se pripisuje Kentu Becku. Ovakav naziv je dobila zato što ideju iterativnog razvoja gura do ekstremnih razina. Ovo je najpopularija i najčešće korištena metodologija razvoja. Razvojni proces se sastoji od kontinuiranog niza brzih iteracija gdje svaka iteracija stavara novu inačicu sustava koja se isporučuje naručitelju. S obzirom da nova inačica najčešće sadrži i novu funkcionalnost XP podsjeća na model inkrementalnog razvoja softvera, no nova inačica može samo popravljati ili mijenjati već postojeću funkcionalnost pa u tom slučaju XP podsjeća na model evolucijskog razvoja. U praksi je to dakle kombinacija inkrementalnog i evolucijskog modela s vrlo kratkim i brzim iteracijama.

Ova metodologija zahtjeve prikazuje kao *user stories*. Na početku iteracije, koja ne bi trebala biti duža od dva tjedna, uzimaju se one priče s višim prioritetom koje se potom razgrađuju u manje zadatke koji se raspoređuju razvojnom timu u skladu sa stupnjem složenosti. Ovaj neformalni postupak planiranja novog izdanja naziva se *planning game*. Korisnici igraju važnu ulogu u XP metodologiji pa zbog toga postoji predstavnik korisnika koji je član razvojnog tima dostupan ostalim članovima. On sudjeluje u kreiranju korisničkih priča, određuje prioritete, definira sadržaj iduće iteracije i testira sustav.

XP je donio nekoliko inovacija kao što je *test-first development* gdje se najprije sastavljaju testovi za novu funkcionalnost, a tek se onda nova funkcionalnost implementira. Pored ovog,

XP donosi praksu programiranja u paru (*pair programming*) gdje jedan zadatak obavljaju dvojica programera koji sjede za istim računalom i zajedno pišu programski kod. Prednost ovakvog načina rada je da dvojica programera u zajedničkoj diskusiji lakše dolaze do kvalitetnog rješenja, a pri tom još jedan drugome ispravljaju pogreške. Također, XP promovira oblikovanje softvera da podrži trenutne zahtjeve jer se smatra da se buduće promjene ionako ne mogu predvidjeti pa se onda i ne isplati o njima brinuti unaprijed. [5], [7], [8], [9]

2.2.4 Scrum

Scrum je agilna metodologija razvoja softvera nastala sredinom 1990-tih godina usmjerena na upravljanje iterativnim procesom što predstavlja ključ Scrum-a. Naziv je preuzet iz ragbija gdje označava trenutak kada se suprostavljeni timovi skupe na gomilu i bore za posjed lopte (osim simbolički, ovo nije vezano za softverski projekt). Kao temelj Scrum-a naglašava se iskustveni pristup i korištenje znanja i prethodnih iskustava, te donošenje odluka na temelju onoga što je poznato. Scrum je zamišljen za rad s malim agilnim timovima od tri do devet članova koji bi trebali imati vrijednosti poput predanosti, hrabrosti, mogućnosti fokusiranja, otvorenosti i poštivanja. Ove timove vodi takozvani Scrum master koji pri tom pazi da se tim pridržava pravila i vrijednosti Scrum-a. On komunicira s korisnicima van tima, ne govori timu kako da odradi posao, on samo upravlja općenitim procesom i daje općenite upute.

Sam proces Scrum-a je jednostavan, a određuje ga *sprint* – vremenski okvir od par tjedana u kojem se izrađuje jedan inkrement iterativnog procesa. Na kraju *sprint-a* se odvija pregled u kojem mogu sudjelovati i korisnici. Nakon pregleda napravi se revizija u kojoj tim iznosi svoje mišljenje, a potom slijedi planiranje sljedećeg *sprinta*. [7], [9], [10], [11]

2.2.5 Dynamic systems development – DSDM

DSDM, slično kao i ostale agilne metodologije, funkcionira po inkrementalnom iterativnom principu uz rad s ljudima. Ipak, razlikuje se od ostalih agilnih metodologija jer kvalitetu i vrijeme stavlja na prvo mjesto pa čak i nauštrb funkcionalnosti. Kod DSDM-a se zahtijeva znatno više dokumentacije nego u ostalim agilnim metodologijama, fokusira se na zahtjeve, propagira isporuku na vrijeme, stalna i jasna komunikacija, suradnja, demonstracija kontrole, inkrementalna izrada, iterativni razvoj i zabrana ugrožavanja kvalitete. Prethodno navedeni principi ostvaruju se sljedećim propisanim tehnikama:

- Time-boxing – sve se MORA odraditi u definiranim vremenskim okvirima,

- MoSCoW – tehnika prioritiziranja rada koja je naziv dobila po: Must, Should, Could, Won´t,
- Prototipiranje,
- Testiranje,
- Modeliranje. [9]

2.2.6 Feature driven development – FDD

FDD je agilna metodologija o kojoj se prvi put javno raspravljalo 1999. u knjizi *Java Modeling in Color with* UML, a koja je prvu primjenu imala na razvoju softvera za financijsku instituciju sa sjedištem u Singapuru s timom od 50 ljudi. To je metodologij koja organizira razvoj softvera oko napredovanja u razvoju funkcionalnosti. Dizajnirana je da prati razvojni proces u pet koraka:

- Razvoj cjelokupnog modela,
- Izrada popisa funkcionalnosti,
- Planiranje po funkcionalnostima,
- Projektiranje prema funkcionalnostima,
- Izrada prema funkcionalnostima.

Slabost ove metodologije očituje se u slabosti kod primjene na manjim projektima. Pored toga, uvelike se oslanja na glavne programere i donošenje odluke odozgo prema dolje, za razliku od npr. XP koji se više temelji na kolektivnom odlučivanju. [7], [9]

2.2.7 Izazovi procjenjivanja u agilnom razvoju

Osnovni izazov kod procjenjivanja je činjenica o prisustvu rizika i neizvjesnosti ishoda. Za kvalitetnu procjenu nužno je imati precizno definirane zahtjeve, a baš to kod agilnog razvoja nije slučaj jer se ne inzistira na dokumentaciji, zahtjevi se mijenjaju i prilagođavaju tokom trajanja iteracija na projektu i ono što bi trebalo biti čvrsto kao ulazna informacija za kvalitetnu procjenu to nije. Upravo zbog toga i nastaju velika odstupanja u odnosu planiranog i realiziranog kod softverskih projekata uz prisustvo agilnog razvoja. Procjene su točnije na manjim projektima što je i logično s obzirom da na manjem projektu imamo i manje elemenata koje trebamo procijeniti za razliku od velikih projekata gdje je tih elemenata više pa se i pogreška u procjeni multiplicira.

Također, važno je znati i što se procjenjuje; vrijeme, novac, veličina tima, kvaliteta ili nešto drugo. U svakom slučaju, sve je ovo međusobno vrlo povezano i isprepleteno i pogreška u jednom segmentu se sigurno očituje i na drugom. Izuzetno je velika povezanost između vremena, novca i kvalitete. Pored toga, neki resursi su nadoknadivi, ali vrijeme ne. Novac možemo nadoknaditi, ali izgubljeno vrijeme nikako ne možemo povratiti.

Pored fukcionalnosti za procjenu je važno znati s kakvim i kolikim razvojnim timom se raspolaže. Znatno je lakše procjenjivati ako su tim i njegove mogućnosti poznate nego ako se radi općenita procjena bez spoznaja o samom timu, a u agilnom razvoju je posebno značajan tim jer on sam često odlučuje o načinu realizacije funkcionalnosti i kontinuirano je uključen u komunikaciju bilo internu ili eksternu. [12]

3 MJERENJE FUNKCIONALNOSTI SOFTVERA

Ako se prihvati izreka da nema ozbiljnog inženjerstva bez točnih i preciznih mjerenja, onda se može reći da je izraz *softversko inženjerstvo* prilično pogrešan jer se od samog početka softverske industrije do danas vrlo teško, rijetko i nevjerodostojno mjere veličina, kvaliteta i produktivnost softvera. Uobičajene softverske metrike poput Linija koda (LOC) su vrlo subjektivne i sa zabrinjavajućom pogreškom. Generalno je vrlo malo toga mjereno u ovoj industriji, a metrike su dvosmislene i subjektivne. Da bi zanat postao znanost i inženjerstvo, nužno je imati točno mjerenje i učinkovite sintetičke metrike. [4]

3.1 Metode mjerenja

Nedostatak mjerenja ne čudi u početcima razvoja ove industrije jer se nije znalo ni što mjeriti ni kako mjeriti. Trebalo je osmisliti, a potom ispitati i potvrditi subjektivne i kvalitetne metode mjerenja. S obzirom da je kompleksnost softvera teško jezično izraziti, nije teško prihvatiti da ju je onda još teže i izmjeriti. Konačno se treba zapitati i što se uopće mjeri? Jesu li to funkcionalnosti, kompleksnost, vrijeme, resursi (novac, ljudi...). sve to zajedno u istoj jednadžbi zaista otežava dolazak do konkretne i subjektivne metode.

Istraživanja pokazuju da samo 30% softverskih tvrtki primjenjuje metode mjerenja na svojim projektima – ali i tada uglavnom u svrhe validacije. [13]

3.1.1 Linije koda kao mjera funkcionalnosti (LOC)

U ranijem razdoblju razvoja softvera, dok on još nije imao grafičko sučelje i dok nije bio poznat pojam objektno orijentiranog programiranja i drugih sličnih pojmova koji su se pojavili tek u kasnijem razdoblju razvoja softvera, programeri su posezali za onim sto su imali i vidjeli pa su softvere koje su pisali iskazivali brojem linija koda (*eng.* Line of Code – LOC) [4]. Prvi zabilježeni spomen pokušaja formalnog mjerenja SW bio je u 70-ima prošlog stoljeća. [14]

LOC metrika i metoda imaju nekoliko bitnih nedostataka. Metoda značajno ovisi o programskom jeziku u kojem se piše SW pa broj LOC ne odražava vjerno funkcionalnost SW. A sam broj LOC je poznat tek po završetku projekta. Nadalje, pouzdanost ove metode je varljiva jer je metrika subjektivna. Naime jedan te isti zadatak dvojica programera će napraviti na različite načine što će rezultirati i različitim brojem linija koda.

3.1.2 Funkcijska točka i na njoj bazirane metode mjerenja funkcionalnosti (FSM)

Nakon mjerenja linijama koda pojavljuje se sredinom 1970-ih nova metrika, razvijena od strane Allana Albrechta i njegovih kolega iz IBM-a i nazvana *funkcijske točke* (*eng*. Function Point [15], [16]. Ova metrika daleko nadilazi metriku LOC jer je općenitija i višekratna, te se može primijeniti i na mjerenje zahtjeva, dizajna, tehničku specifikaciju, kodiranje i slično. Ona je osnova za primjenu metode *mjerenja funkcionalnosti funkcijskim točkama* (*eng. Functional Size Measurement – FSM* ili *Function point Analysis – FPA*).

FPA omogućuje mjerenje funkcionalnosti SW iz perspektive korisnika [15], [4]. Ova metoda se zasniva na 5 vidljivih i dobro definiranih eksternih karakteristika SW [4], [16], [17], [18], [19], [20], [21]:

1. **podatkovni entiteti:**
   - interni logički fileovi (*eng. Internal Logical Files* – ILF) – podaci koji ostaju unutar SW;
   - externi podaci za razmjenu (*eng. External Interface File* – EIF) – podaci koji se razmjenju s drugim sustavima.
2. **transakcije:**
   - eksterni ulazi (*eng. External Inputs*) u SW – logičke ulazne transakcije;
   - eksterni izlazi (*eng. External Outputs*) ili eksterni upiti (*eng. External Queries*) – npr. online prikazi, izvješća ili slanja u druge sustave.

Metode bazirane na funkcijskoj točki kao mjernoj jedinici su tijekom godina evoluirale i postale široko prihvaćene metode mjerenja funkcionalnosti softvera te je 1986. godine osnovana i stručna udruga International Function Point Users Group (IFPUG). Naknadno je IFPUG Function Points metoda prihvaćena i kao međunarodni ISO standard za mjerenje funkcionalnosti softvera – ISO/IEC 20926 (zadnja revizija je bila 2019. godine) [22].

3.1.3 Divergencija metoda funkcijskih točaka

Tijekom godina došlo je i do divergencije metoda baziranih na funkcijskim točkama tako da ima više od 20 varijanti od kojih je pet potvrđeno od strane međunarodne organizacije ISO [4], [23]:

- IFPUG FPA, izvorna Albrechtova (sadašnja poznata kao verzija IFPUG 4.1) i sukladna s ISO/IEC 20926:2009
- Mark II FPA, sukladna s ISO/IEC 20968:2002
- NESMA FPA, kreirala ju Nizozemska udruga za softversku metriku (Netherlands Software Metric Association) i sukladna s ISO/IEC 24570:2005
- FiSMA FSM, kreirala ju Finska udruga za softversku metriku (Finland Software Metric Association) i sukladna s ISO/IEC 29881:2008 i najnovija
- COSMIC, kreirao ju COmmon Software Measurement International Consortium.

Temeljne razlike među ovim varijantama metoda mjerenja funkcionalnosti softvera proizlaze iz sljedećih činjenica [24]:

- metode se baziraju na vlastitim definicijama funkcionalnosti,
- metode se koriste različitim pravilima bojama/mjerenjima za različite funkcionalne komponente funkcijskih zahtjeva korisnika ili sâmog softvera
- metode imaju vlastite mjerne jedinice i skale za svoje mjere.

Ovo mnoštvo varijanti također je jedna od zapreka širem korištenju softverskih metrika i metoda mjerenja funkcionalnosti softvera [20]. Postoje čak i aplikacije za automatizirano mjerenje veličine funkcijskim točkama. Međutim, njihova točnost je upitna. [25]

3.1.4 Objektne metrike i metode mjerenja funkcionalnosti softvera

S primjenom objektno orijentiranog programiranja i agilnih metodologija razvoja softvera počele su se primjenjivati nove prakse i novi artefakti. Sa stajališta metrika i metoda mjerenja posebno važni postali su *use case* (*studija slučaja*) i *user story* (*korisnička priča*), kao nešto što se može jasno identificirati, pratiti pa stoga i mjeriti. Na njima se temelje najčešće korištene objektne mjerne jedinice za mjerenje/izražavanje funkcionalnosti i kompleksnosti softvera [4], [20], [26], [27]:

- Use Case Point (UCP) i
- Story Point (SP).

UCP metoda je inovacija Gustava Karnera iz 1993. godine [14]. U ovoj metodi razlikuju se i broje dva temeljna elementa ovog UML dijagrama:

- sudionici slučaja i
- transakcije koje su nužne da bi se taj slučaj dogodio.

Metoda opisuje *funkcionalnosti* koje softver treba imati a ne *način* kako će one biti implementirane. Za razliku od funkcijskih točaka UCP može pokriti širi dijapazon tipova aplikacija. Međutim kod UCP metoda nedostaju metode međusobne konverzije i standardizacija kakva postoji kod metoda baziranih na funkcijskim točkama. [4], [14], [20], [28]

SP je mjerna jedinica za mjerenje veličine ukupne korisničke priče, dijela korisničke priče (tj. samo neke funkcionalnosti) ili neke druge jedinice rada na projektu. Apsolutna vrijednost koja se pritom dodjeljuje priči nije presudna – bitan je relativan odnos vrijednosti koji se dodjeljuje pričama kako bi se one mogle međusobno uspoređivati. [27]

3.2 Algoritamske metode

Algoritamske metode su metode temeljene na matematičkim algoritmima ili parametarskim jednadžbama, čiji rezultat je obično potreban da bi se dobila dozvola za nastavak projekta, te su uključeni u poslovne planove, budžet, te ostala financijska planiranja i mehanizme za praćenje razvoja tog istog softvera [29]. Obično su se ti algoritmi izvodili ručno, ali u današnje vrijeme su gotovo univerzalno kompjuterizirani. Mogu biti standardizirani (dostupni u objavljenim tekstovima ili kupljeni), ili vlasnički, ovisno o vrsti poslovanja, proizvodu ili projektu. Algoritamske metode se zasnivaju na matematičkim modelima koji proizvode procjenu troška kao funkciju broja varijabli koje se smatraju najvećim faktorima troška. Svaki algoritamski model ima formu:

$$\text{Uloženi\_trud} = f(x_1, x_2, \ldots, x_n), \tag{3.1}$$

gdje su $\{ x_1, x_2, \ldots, x_n)$ faktori troška.

Postojeće algoritamske metode se razlikuju u dva aspekta: u selekciji faktora troška, i u formi funkcije f.

3.2.1 Faktori troška

Osim veličine softvera, postoje i mnogi drugi faktori troška. Ti troškovni čimbenici mogu se podijeliti u četiri osnovne vrste:

1) **Čimbenici proizvoda:** potrebna pouzdanost, kompleksnost proizvoda, veličina korištene baze podataka, mogućnost višekratnog korištenja, dokumentacija koja odgovara potrebama životnog ciklusa proizvoda,
2) **Računalni čimbenici:** vremensko ograničenje izvršavanja, ograničenje glavnog skladišta za pohranu, nestalnost platforme,
3) **Čimbenici osoblja:** analitička sposobnost, iskustvo u radu s aplikacijom, sposobnost programiranja, iskustvo na platformi, iskustvo u radu s programskim jezikom i alatima, kontinuitet osoblja,
4) **Čimbenici projekta:** razvoj više stranica, korištenje softverskog alata, potreban kalendar razvoja.

Ovi faktori nisu nužno neovisni, dapače, u mnogim modelima neki od faktora se pojavljuju u kombiniranom obliku, a neki se jednostavno ignoriraju.

3.2.2 Linearni i multiplikativni modeli

Linearni modeli imaju oblik:

$$\text{Uloženi\_trud} = a_0 + \sum_{i=1}^{n} a_i x_i \tag{3.2}$$

gdje su koeficijenti a_0, …, a_n odabrani da najbolje odgovaraju podacima završenog projekta [29].

Multiplikativni modeli imaju oblik:

$$\text{Uloženi\_trud} = a_0 + \prod_{i=1}^{n} a_i^{xi} \tag{3.3}$$

gdje su opet koeficijenti a_0, …, a_n odabrani da najbolje odgovaraju podacima završenog projekta. Walston-Felix je koristio ovakav tip modela gdje je svaki x_i koristio samo tri moguće varijable: -1, 0, +1. Doty model također pripada ovoj klasi gdje svaki x_i može zauzimati samo dvije vrijednosti: 0, +1.

3.2.3 Funkcijski modeli

Funkcijski modeli imaju opći oblik:

$$\text{Uloženi\_trud} = a \times S^b \times M, \tag{3.4}$$

gdje je *S* veličina koda, a *a* i *b* su obično jednostavne funkcije drugih čimbenika troška, *M* je množitelj dobiven kombiniranjem procesa, proizvoda i atributa razvoja. Dva najpopularnija algoritamska modela koja se koriste su COCOMO i Putnam [26].

3.2.3.1 COCOMO (Constructive Cost Model) modeli

Ovu obitelj modela predložio je Boehm, te su naširoko prihvaćeni u praksi [29]. U COCOMO modelima veličina koda *S* se izražava u tisućama LOC (KLOC) – linijama koda, a uloženi trud u čovjek/mjesecu. COCOMO modeli su jako dobro dokumentirani, dostupni u javnoj domeni i podržani od strane javne domene i komercijalnih alata, ima dugu povijest od svog prvog korištenja 1981. do najnovije verzije, COCOMO II, izdane 2000. [29], [30], [31]

Postoje tri podjele COCOMO modela:

1) Osnovni COCOMO,
2) Srednji COCOMO model i detaljni COCOMO,
3) COCOMO II.

3.2.3.2 Putnam model i SLIM

Putnam izvodi svoj model temeljen na Norden/Rayleigh distribuciji radne snage i njegovim pronalascima u mnogim završenim projektima. Centralni dio Putnamovog modela se naziva *softverska jednadžba* koja glasi:

$$S = E \; x \; (Uloženi\_trud)^{1/3} \; t_d^{4/3}, \tag{3.5}$$

gdje je t_d vrijeme isporuke softvera, E je faktor okoliša koji odražava sposobnost razvoja, a koji se može dobiti iz povijesnih podataka koristeći softversku jednadžbu. Veličina S je izražena u LOC (lines of code – linije koda), i Uloženi trud u čovjek/godini. Još jedna važna relacija koju je postavio Putnam je:

$$\text{Uloženi\_trud} = D_0 \text{ x } t_d^3 , \tag{3.6}$$

gdje je D_0 parametar nazvan izgradnja radne snage, koji se kreće od vrijednosti 8 (potpuno novi softver sa mnogo sučelja), do 27 (ponovno izgrađeni softver). Kombiniranjem gornje jednadžbe sa softverskom jednadžbom dobiva se forma funkcije:

$$\text{Uloženi\_trud} = (D_0^{4/7} \text{ x } E^{-9/7}) \text{ x } S^{9/7} \text{ i} \tag{3.7}$$

$$t_d = (D_0^{-1/7} \text{ x } E^{-3/7}) \text{ x } S^{3/7} \quad (3.8)$$

Putnam model se također jako koristi u praksi, a SLIM je softverski alat temeljen na ovom modelu za procjenu troškova softvera i raspoređivanja radne snage. [26]

3.2.4 Diskretne i ostale metode

Diskretne metode imaju tablični oblik, koji obično izražava trud, trajanje, težinu i druge čimbenike troška. Ova klasa modela sadrži Aron model, Wolverton model i Boeing model. Ovi su modeli u početku razvoja procjene troškova dobili na popularnosti zbog iznimno lakog rukovanja. Od ostalih modela poznat je Price-S model razvijen od strane RCA, New Jersey, te SoftCost. [29]

3.3 Problemi mjerenja i razlog za procjenjivanje

Prethodno je konstatirano da postoji jako malo mjerenja na softverskim projektima (*v. poglavlje 3.1*), te da je nužno postojanje metrika kako bi zanat prerastao u inženjerstvo. No, da bi se moglo mjeriti, potrebno je imati gotov softver, a onda se postavlja pitanje potrebe za mjerenjem nečeg što je već napravljeno. Vjerojatno je i to jedan od razloga zbog kojeg nedostaju podatci o ranije realiziranim projektima. Ti podatci mogu pomoći kod statističkih poređenja sličnih projekata pa se njihov nedostatak može osjetiti u tom pogledu.

S druge strane, softver najčešće ima svog naručitelja koji prije ulaska u projekt želi znati koliko će projekt zahtijevati resursa bilo u vremenu ili novcu. Mjerenje u tako ranoj fazi projekta ne može se obaviti jer se još uvijek nema *što* mjeriti i tu se dolazi do potrebe za procjenom. No i za procjenu su potrebni neki ulazni parametri pa se dolazi do svojevrsne petlje. Ulazni podatci bi trebali proizići iz specifikacije zahtjeva, a s obzirom na sve učestalije korištenje agilnih metoda u razvoju softvera koje zamjenjuju one klasične, dokumentacija je sve oskudnija. Pogotovo u ranoj fazi projekta, a onda je procjena važna na samom početku ili čak prije njega. Dakle, kod inicijalizacije projekta važno je imati precizne ulazne informacije kako bi se korištenjem neke od metoda procjenjivanja došlo do procjene na temelju koja će iskazati potrebne resurse za realizaciju projekta. Po okončanju projekta bilo bi uputno izmjeriti ono što se na početku procjenjivalo kako bi se to usporedilo i vidjelo u kojoj mjeri se procjena razlikuje od mjerenja. Takve usporedbe, a pogovotu s povećanjem njihove količine, mogu biti dobra baza znanja za sve buduće procjene ali i za definiranje pouzdanijih metoda procjenjivanja. One omogućuju povratnu informaciju o cijelom procesu razvoja i životnom

ciklusu projekta koji je bio predmet procjene. Neki ovu uzročno posljedičnu vezu nazivaju paradoksom jer se ne može procijeniti bez nekog mjerenja, a kad se na kraju ima rezultate mjerenja onda ta informacija više nije potrebna [32].

4 METODE PROCJENJIVANJA

4.1 Procjenjivane varijable

Točna procjena vremena i troška za razvoj softverskih projekata ozbiljan je problem za softverske inženjere [33]. Stalno korištenje jasno definiranih i dobro razumljivih procesa razvoja softvera pokazalo se najefikasnijom metodom dobivanja podataka za statističku procjenu, na temelju čega su se razvile različite metode za procjenu.

Značajan faktor kod procjenjivanja projekta je njegova veličina. Manji projekti[1] teško mogu imati osobu koja će aktivno koristiti statistički orijentirane tehnike te su prevelika odstupanja zbog izražene individualnosti i utjecaja pojedinca na sam projekt. Za veće projekte, gdje je uključeno više od 20 ljudi i koji traju više od 6 mjeseci, individualnost pojedinca se u većoj mjeri uklapa u statističke prosjeke timova te zbog toga manje dolazi do izražaja.

S obzirom na razvojni stil definiranja zahtjeva razlikuje se sekvencijalnu i iterativnu tehniku. Kod sekvencijalnog definiranja zahtjeva na samom početku projekta u njegovoj prvoj fazi od definiranja koncepta teži se da većina zahtjeva bude definirana i poznata, te je ova faza značajno duža u usporedbi s iterativnim projektima gdje je ona kraća i prije se kreće u ranu izgradnju gdje se zahtjevi definiraju tokom cijelog trajanja projekta iz iteracije u iteraciju. Sve te varijacije utječu na korištene metode procjene s obzirom na njihovu točnost u procjenjivanju ali i na cijenu samog procjenjivanja.

Napor je najveća i najnepredvidljivija komponenta ukupnih troškova projekta kod razvoja softvera pa se stoga najviše pažnje usmjerava upravo u njegovo predviđanje. Postoje algoritamske i nealgoritamske metode procjene. Nealgoritamske se temelje na vlastitim iskustvima stručnjaka koji su već radili na takvim projektima i imaju dovoljno znanja, na vlastitim procjenama za svaki dio softvera posebno ili uspoređivanjem sa nekim sličnim projektom. Algoritamske metode temelje se na matematičkim algoritmima i parametarskim jednadžbama za dobivanje željenog rezultata. [29], [34]

[1] Njih karakterizira manji broj osoba u timu (najčešće 5-10 članova) i trajanje od pâr mjeseci.

4.2 Nealgoritamske metode

Nealgoritamske metode procjene resursa razvoja softvera temelje se na prethodnim procjenama razvoja softvera, na temelju iskustava stručnjaka koji su već radili na takvim ili sličnim projektima, ili analizirajući dio po dio projekta i donošenjem procjena. Neke od tih metoda su procjena prema povijesnoj analogiji, procjena stručnjaka, "top-down" metoda, "bottom-up", "price-to-win" metoda itd. [29]

4.2.1 Procjena po analogiji

Metoda procjene po povijesnoj analogiji zahtijeva jedan ili više završenih projekata razvoja koji su slični novom projektu, te se izvodi prema analogiji koristeći stvarne troškove prethodnih projekata, njihovu veličinu ili uloženi trud. Može se obavljati na razini ukupnog projekta, ili na razini podsustava. Prednost koje ima procjena na razini ukupnog projekta je ta što se razmatraju sve komponente sustava, dok je prednost procjene na razini podsustava ta što se pružaju detaljnije procjene sličnosti i razlike između novog projekta i završenih projekata. Snaga procjene temeljene na analogiji je ta što se temelji na stvarnim projektim iskustvima. Da bi se dobro provela ova metoda, mora se proći 5 osnovnih koraka [29]:

1) Dobivanje detalja o veličini, uloženom trudu, i troškovima za sličan prijašnji projekt,
2) Usporediti veličinu novog projekta dio po dio sa starim projektom,
3) Izraditi procjenu veličine novog projekta kao postotak veličine starog projekta,
4) Izraditi procjenu resursa koristeći procjenu iz točke 3,
5) Provjeravati stalne pretpostavke kroz stari i novi projekt.

4.2.2 Procjena stručnjaka

Ova metoda procjene uključuje savjetovanje sa jednim ili više stručnjaka, na način da se procjena radi na temelju onoga što se procjenitelj sjeća da je bilo potrebno za izvršavanje prijašnjeg sličnog projekta, te veličine samog projekta [29]. To je obično subjektivna procjena na temelju vlastitih metoda i iskustva stručnjaka, te se ta procjena uzima kao relativno točna ako je stručnjak procjenitelj imao nedavno iskustvo u razvoju oba dijela samoga projekta, kako u softverskom, tako i u procesu same procjene. Kada isti sustav procjenjuje više stručnjaka, koriste se mehanizmi konsenzusa stručnjaka kao što su Delphi tehnika ili PERT da bi se riješile nedosljednosti u procjeni. Delphi tehnika funkcionira na sljedeći način [29]:

1) Koordinator predstavlja svakom stručnjaku specifikacije i obrazac za evidenciju procjene,
2) Svaki stručnjak ispunjava obrazac individualno, te je dozvoljeno postavljati koordinatoru pitanja,
3) Koordinator priprema sažetak svih procjena na obrascu, te zahtijeva još jedno ponavljanje procjena uz obrazloženje,
4) Ponavljanje koraka 2. i 3. koliko je potrebno.

Prema knjizi "Discourse Analysis as Theory and Method" pojedinačna stručna prosudba daleko je najčešći pristup procjene koji se koristi u praksi [35]. U literaturi [36] navodi se da 83% procjenitelja koristi "neformalnu analogiju" kao svoju primarnu tehniku procjene. Autori istraživanja [37] otkrili su da se 72% procjena projekta temelji na "stručnom mišljenju". Stoga se prema navedenim podacima može zaključiti da ekspertne procjene spadaju među najviše korištene metode procjenjivanja.

U literaturi [35] napominje se da povećano iskustvo u aktivnosti koja se procjenjuje ne dovodi nužno do povećane točnosti u procjeni same aktivnosti. Biti stručnjak za tehnologiju ili razvojne prakse i alate koji se koriste ne podrazumijeva da ćete biti stručnjak za procjenu istog. A u istraživanju [38] autori tvrde da najtočnije procjene određenih zadataka poput vremena potrebnog za kodiranje neke određene funkcionalnosti mogu dati ljudi koji će zaista i obaviti taj posao. Ili drugim riječima, procjene koje donose ljudi koji ne rade taj posao su manje točne i vjerojatnije je da će se potcijeniti ako procjenu vrše procijenitelji koji nisu vezani uz projekt nego samo osobe koje razvijaju predmetni softver.

4.2.3 Parkinson metoda

Koristeći Parkinsonov princip "rad se širi da popuni volumen na raspolaganju", resursi su određeni (nisu procijenjeni) raspoloživim resursima, a ne na temelju objektivne procjene. Ako softver mora biti isporučen u 12 mjeseci, a dostupno je 5 osoba, trud se procjenjuje na 60 čovjek/mjeseci. Iako to ponekad daje dobre procjene, ova metoda se ne preporuča jer može pružiti vrlo nerealne procjene. Također, ova metoda ne promovira dobre prakse s područja programskog inženjerstva. [29]

4.2.4 „Price-to-win“ metoda

Trošak razvoja softvera se procjenjuje tako da najbolja cijena pobijedi. Procjena se temelji na proračunu kupca umjesto na funkcionalnosti softvera. Na primjer, ako razumna procjena troška projekta iznosi 100 čovjek/mjeseci, ali kupac može priuštiti najviše 60 čovjek/mjeseci, uobičajeno je da se zatraži od procjenitelja da prilagodi procjenu prema proračunu koji ima kupac, da bi dobio taj projekt. Mana ovakvog načina procjene je što će se vjerojatno prouzročiti kašnjenje isporuke projekta ili će se prisiliti razvojni tim na prekovremeni rad. [29]

4.2.5 „Bottom-up“ i „Top-down“ metode

U "Bottom-up" metodi svaka komponenta softverskog sustava je zasebno procijenjena te se rezultati spajaju i daju procjenu za ukupni sustav. Zahtjevi za ovakav pristup su ti da se početni dizajn mora napraviti takav da se lako razloži u različite komponente. "Top-down" metoda je suprotna "Bottom-up" metodi. Opća procjena troškova razvoja sustava izvedena je iz općih svojstava sustava, koristeći se algoritamskim ili ne-algoritamskim metodama procjene. Ukupni trošak se tada može podijeliti među različite komponente. Ovakav pristup procjeni je prikladniji za procjenu troškova u ranoj fazi projekta. [29]

5 PROBLEMI I IZAZOVI PROCJENJIVANJA

5.1 (Ne)uspješnost softverskih projekata

Surova ali istinita činjenica je da softverski projekti propadaju. Propadaju iz više razloga, a jedan od njih je sigurno i procjenjivanje projekata prije ugovaranja istih. Projekt ugovoren na pogrešnoj procjeni najvjerojatnije ima krivu pretpostavku o vremenu i novcu potrebnom za njegovo izvršenje. Iz ovog je više nego očito, da iako je planiranje i procjenjivanje kompleksno i najčešće pogrešno, ono je ipak ključno za uspjeh softverskih projekata.

U prilog tezi da je procjenjivanje često pogrešno idu i sljedeći podatci koji kažu da se tek 29% softverskih projekata završi uspješno. U ovom podatku je zabrinjavajuća činjenica da je ovaj isti postotak uspješnosti prisutan u statistikama posljednjih 15 godina usprkos radu struke na unaprijeđenju načina i metodologija razvoja i procjenjivanja [39]. Sudbinu neuspješnih projekata nosi 19% njih, a čak 52% projekata završi s poteškoćama kao što su prekoračen rok, prekoračen budžet, smanjen broj isporučenih funkcionalnosti od ugovorenih i slično. Tablica koja slijedi daje statistički pregled projekata s obzirom na uspjeh/neuspjeh u razdoblju od 2011. do 2015. godine.

Tablica 5-1 Pregled uspješnosti projekata [39]

	2011	**2012**	**2013**	**2014**	**2015**
Uspješni	29%	27%	31%	28%	29%
Završeni s poteškoćama	49%	56%	50%	55%	52%
Neuspješni	22%	17%	19%	17%	19%

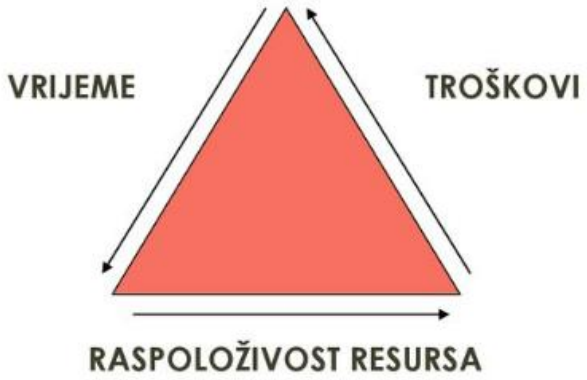
Sljedeća tablica daje pregled uspješnosti projekata s obzirom na veličinu projekta. Primjetno je da su najuspješniji manji projekti što je i očekivano s obzirom da je velike i mega projekte vrlo teško procjenjivati.

Tablica 5-2 Pregled uspješnosti projekata s obzirom na veličinu [39]

	Uspješni	Završeni s poteškoćama	Neuspješni
Mega projekti	2%	7%	17%
Veliki projekti	6%	17%	24%
Umjereno veliki projekti	9%	26%	31%
Srednji projekti	21%	32%	17%
Mali projekti	62%	32%	17%

5.2 Predmet procjene: funkcionalnost ili resursi?

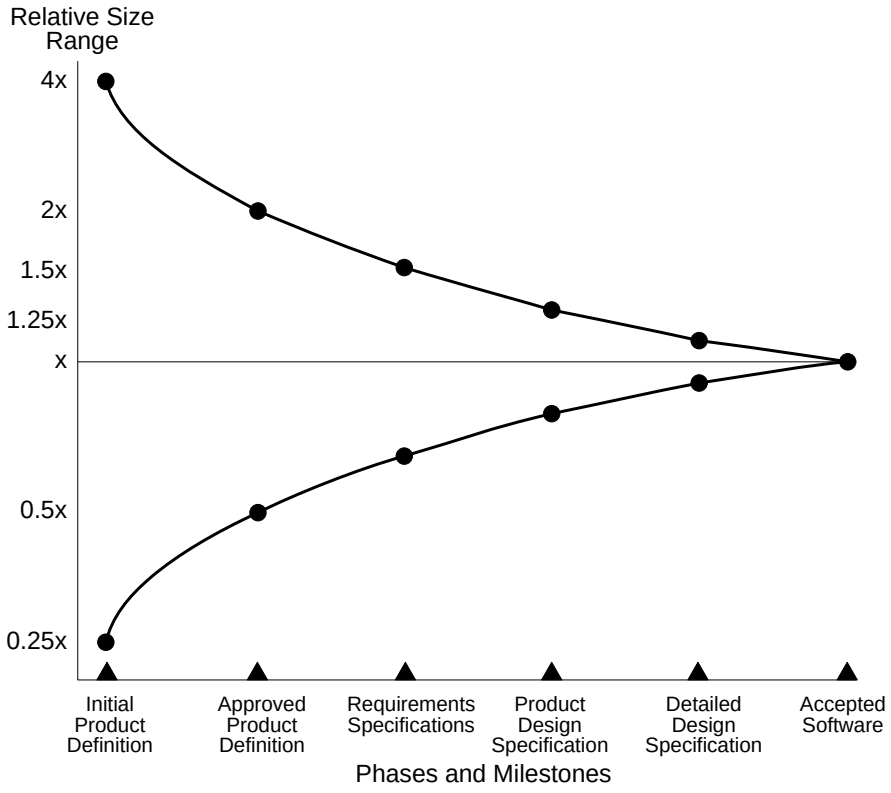
Dobra procjena zahtijeva iskustvo, znanje, pristup informacijama iz prošlih projekata i nadasve hrabrost za obvezu na kvantitativna predviđanja kada su kvalitativne informacije sve što vam je dostupno. Čim se radi o procjeni, automatski je prisutan rizik i neizvjesnost ishoda. Različite metode procjenjivanja primjenjuju se u različitim okolnostima, pa je glavno pitanje pri odabiru metode što se procjenjuje. Ponekad se procjene vrše na temelju zadanih novčanih sredstava i vremenskog okvira pa se procjenjuje što je unutar tog budžeta i vremena moguće isporučiti, a ponekad se definiraju funkcionalnosti i zahtjevi softvera na osnovu čega se procjenjuje potrebno vrijeme i napor za isporuku definiranog. Prethodno navedeno je usko povezano pa to znači da varijabilnost u softverskim zahtjevima za performansama i funkcionalnostima nužno znači i nestabilnost u troškovima, odnosno, u potrebnom vremenu i naporu koji se u konačnici preliju u troškove. Slično tome, nerealno postavljeni vremenski okvir kada su napori potcijenjeni nužno će za posljedicu imati lošiju kvalitetu i/ili manji broj isporučnih funkcionalnosti.



Slika 5-1 Trokut resursa [40]

5.3 Konus nesigurnosti

Nesigurnost u procjeni nije jedanaka u svim životnim fazama softverskog projekta. Svaka od mnoštva odluka i koraka tokom životnog ciklusa razvoja softvera nosi svoj teret nesigurnosti. Pa iako nesigurnost jedne odluke može biti mala, kumulativno dolazi do velikih nesigurnosti. No, kako se s vremenom povećava postotak donesenih odluka, smanjuje se neizvjesnost ukupne procjene. Istraživanjima se zaključilo da je razina nesigurnosti ipak predvidiva po fazama razvoja. U samom početku planiranja projekta postoji najviše nesigurnosti, a kako projekt napreduje smanjuje se nesigurnost kao i broj odluka koje treba donijeti. Boehm je prvi prikazao graf prikazan na sljedećoj slici, a McConnell ga je nazvao konus nesigurnosti (*eng. Cone of unceratainty*) (Slika 5-2).



Slika 5-2 Konus nepouzdanosti [14], [27], [41]

Horizontalna os predstavlja faze razvoja kroz vrijeme, a vertikalna varijabilnost u procjeni općenito (napora, funkcionalnosti, vremena itd.). Najčešće se govori o opsegu projekta koji je preko trokuta resursa povezan sa svim ostalim resursima.

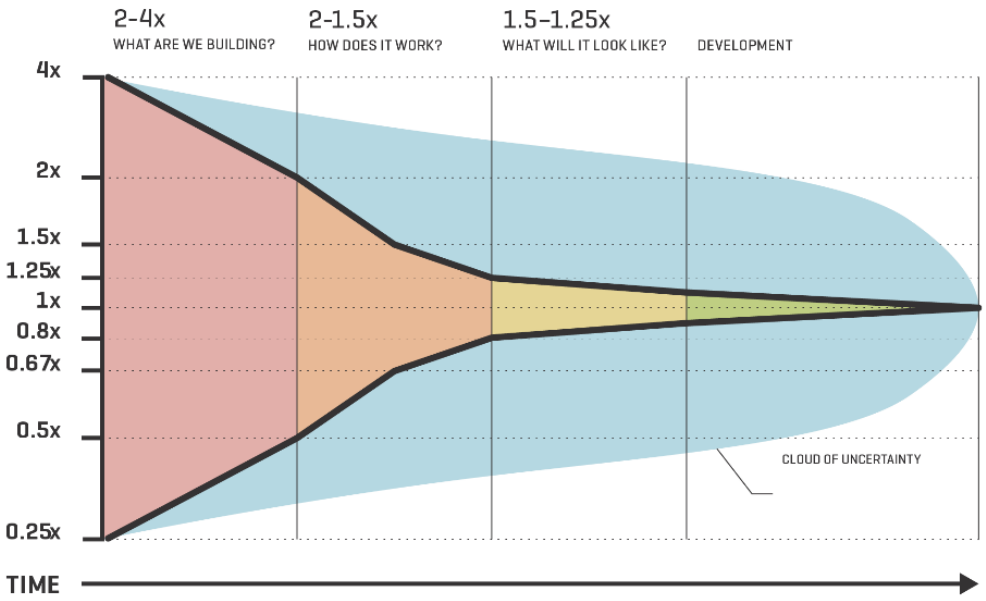
Procjene na samom početku projekta mogu imati visoku grešku, od 4 puta potcijenjenog objuma (0.25x) do 4 puta precijenjenog obujma (4x). Ovdje se govori o greškama iskusnih osoba, a manje iskusne mogu imati i mnogo veću grešku. Bolje rezultate od konusa moguće je dobiti samo sretnim pogađanjem. U procjenama se može koristiti konus nesigurnosti da bi izrazio opseg procjene. Primjerice, ako se radi procjena na samom početku projekta, procjena bi trebala biti opseg, a ne jedna točka. Razlika između najboljeg slučaja i najlošijeg slučaja bi mogla biti čak 16 puta na samom početku projekta, tijekom pisanja inicijalnog koncepta. Kako projekt napreduje, varijabilnost se smanjuje, te su sljedeće procjene preciznije i imaju manji opseg.

Konus nesigurnosti ukazuje na to da vrijeme utrošeno na procjenjivanje ne može utjecati na preciznost procjene. Istraživanje tvrtke Luiz Laranjeira sugerira da točnost procjene softvera ovisi o razini preciziranja definicije softvera. Što je definicija točnija, točnija je procjena. Razlog zbog kojeg procjena sadrži varijabilnost jest taj da sâm softverski projekt sadrži varijabilnost. Jedini način da se smanji varijabilnost u procjeni jest smanjiti varijabilnost u projektu [42].

5.3.1 Oblak nesigurnosti

Važno je shvatiti da se konus ne sužava sam po sebi. Bitno je u svakoj fazi što više smanjiti varijabilnost jer se time smanjuje i greška u procjeni. U suprotnom dolazi do problema kod projekata:

- koji se ne fokusiraju na smanjenje varijabilnosti,
- loše se kontroliraju i
- rijetko procjenjuju.

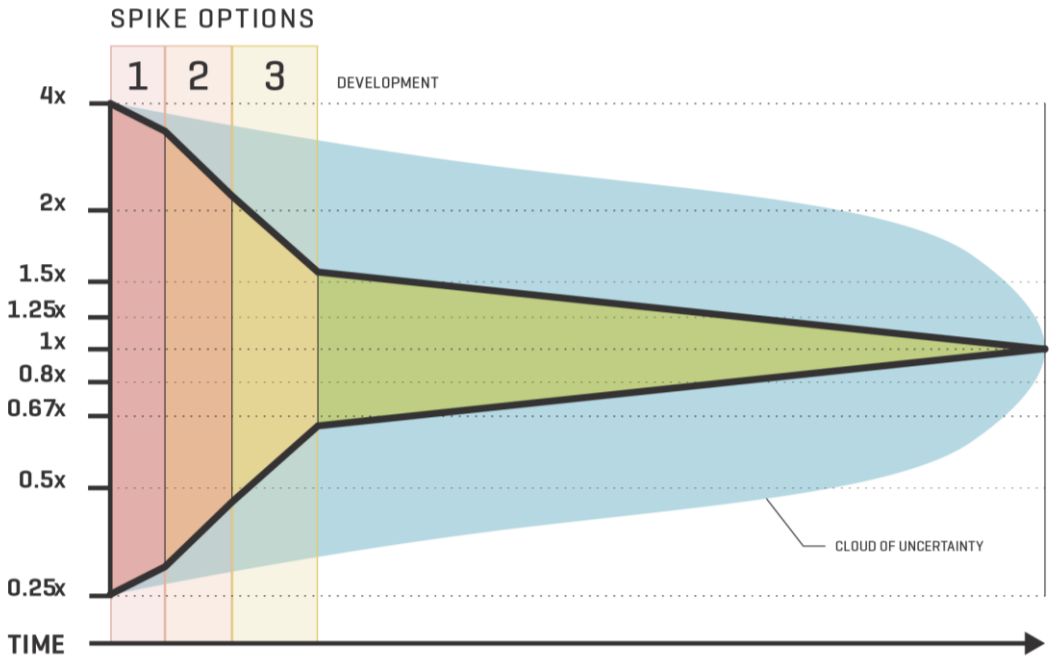


Slika 5-3 Oblak nesigurnosti [43]

Što je više stvari definirano, to je više smanjena i varijabilnost, a posljedično dolazi do najvećeg smanjenja konusa u tim fazama definiranja projekta kada se definiraju zahtjevi, korisničko sučelje, arhitektura i sl.

5.3.2 Konus nesigurnosti u agilnom razvoju

Pravilna primjena agilnog razvoja u suštini se bazira na brže sužavanje konusa. Za razliku od vodopadnog (*eng. Waterfall*) razvoja koji pokušava definirati sve zahtjeve prije razvoja, u agilnom razvoju se kreće od najvažnijih odluka poput one da je *projekt krenuo*. Glavni i osnovni problem vodopadnog modela je skriven u zaista teškom definiranju svih zahtjeva *unaprijed* od strane projektnog tima, ali i investitora/klijenta. Budući da se agilni razvoj događa u iteracijama, svaka od njih se može promatrati kao jedan mali konus nesigurnosti (Slika 5-4).



Slika 5-4 Konus nesigurnosti u agilnom razvoju [43]

5.4 Primjena strojnog učenja i umjetne inteligencije u procjenjivanju

5.4.1 Renesansa 'starih' metoda

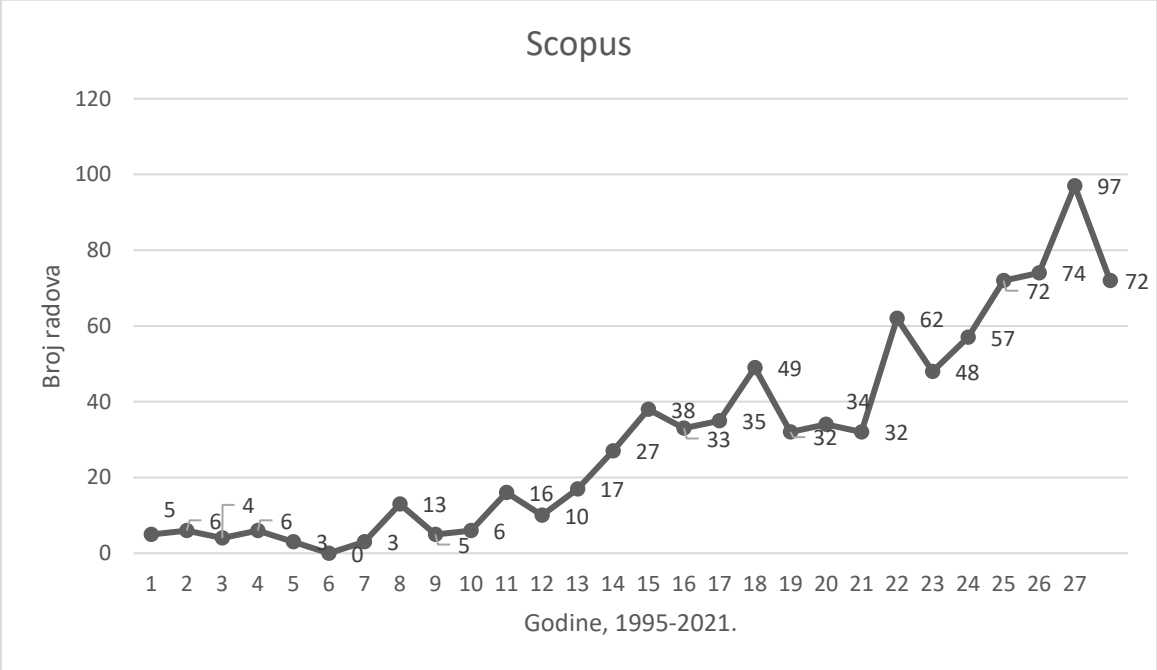
Interes za procjenjivanje ne prestaje jer potreba za pouzdanim procjenama stalno postoji. [44], [45] S druge strane, kontinuirani napredak algoritama i tehnika strojnog učenja i umjetne inteligencije te opći tehnološki napredak omogućili su njihovu djelotvornu primjenu u procjenjivanju parametara bitnih za razne vrste projekata – pa tako i u procjenjivanju softverskih projekata. [28], [46], [47] [48] Zahvaljujući tome svoju renesansu doživljavaju 'stare' metode mjerenja kompleksnosti i funkcionalnosti softvera, koje se sada koriste isključivo u svrhu procjenjivanja. Interes je povećan i za algoritamske i za nealgoritamske metode, opisane u prethodnim poglavljima (*v. poglavlje 3 i poglavlje 4*). [49]

Pretraživanje znanstvene literature u bazi Scopus također pokazuje povećan interes primjene algoritama strojnog učenja i umjetne inteligencije u svrhu *procjenjivanja* na softverskim projektima. Pretraženi su znanstveni radovi iz područja *računarstva* (radovi iz

znanstvenih časopisa i zbornika znanstvenih skupova) koji u naslovu ili sažetku ili ključnim riječima sadrže sljedeće pojmove[2]:

- *"Artificial Intelligence" ILI "Machine Learning" ILI "Neural Network" ILI "Bayes Network"* I
- *"Effort Estimation" ILI "Software Cost Estimation" ILI "Cost Estimation".*

Iz rezultata prikazanih na donjoj slici jasno je uočljiv trend porasta interesa znanstvenika i istraživača za primjenu AI i ML za rješavanje problematike procjenjivanja na projektima razvoja softvera.



Slika 5-5 Broj Scopus radova s primjenama AI, ML, BN ili NN za 'procjene' u 'računarstvu'

[2] Točan upit:
TITLE-ABS-KEY(("Artificial Intelligence" OR "Machine Learning" OR "Neural Network" OR "Bayes Network") AND ("Effort Estimation" OR "Software Cost Estimation" OR "Cost Estimation")) AND (LIMIT-TO (SUBJAREA,"COMP"))

5.4.2 Razvoj novih modela, metoda i sustava za procjenjivanje

Tehnike i algoritmi strojnog učenja i umjetne inteligencije omogućuju unaprijeđenje različitih klasičnih, 'starih' metoda procjenjivanja (uključujući i COCOMO varijante) i izradu novih modela za procjenjivanje u razvoju softverskih projekata. Istraživači se nisu zadržali ovdje, otišli su i dalje u svojim istraživanjima – čak se mogu naći i primjeri razvoja novih sustava (aplikacije) koji se koriste za automatizirane procjene na softverskim projektima. [47], [48]

Istraživanja pokazuju da procjene napora skoro polovice agilnih timova imaju pogrešku od najmanje 25%. [50] Tehnike strojnog učenja, temeljene na algoritmima umjetne inteligencije, omogućavaju učenje na prethodno završenim projektima i automatsko učenje na temelju tih iskustava bez eksplicitnog programiranja, pa su pogodan alat za povećanje uspješnosti procjena na softverskim projektima.

Na ovaj način se primjenom ML i AI u procjenjvanju nastoji nadomjestiti ono što su procjenjivanju 'oduzele' agilne metodologije razvoja softvera – artefakte koje se moglo mjeriti te na osnovu njih procjenjivati elemente softverskih projekata. Kao mjera točnosti ovih novih modelâ i metodâ za procjenjivanje često se koristi *relativna pogreška srednje veličine* (*eng. mean magnitude relative error – MMRE*). [47], [50], [51], [52]

6 ZAKLJUČAK

Procjena se uglavnom temelji na subjektivnim mišljenjima ili intuiciji pa je shodno tome i odluka nastala na temelju procjene vrlo često subjektivna. Procjena u poslovnom smislu predstavlja ogroman izazov jer loše procjenjivanje može uzrokovati propast projekta ili još gore čitavog poslovanja. Stoga se kontinuirano teži smanjenju apstrakcije i povećanju točnosti procjene s egzaktnim ulaznim informacijama i tehnikama što iziskuje stalne napore u istraživanju metoda procjenjivanja.

Procjena i planiranje su vrlo povezani iako između njih postoji temeljna razlika. Procjena je nepristran analitički proces dok je s druge strane planiranje pristran proces koji rezultira konkretnim ciljevima. Procjena teži točnosti, a planiranje teži ostvarivanju rezultata. Njihova očita povezanost ogledava se u tome što se planiranje radi na temelju procjene. Kada se još u prethodno navedeno uključi naručitelj/klijent, nastaje ugovorna obveza. Njima najčešće prethodi ponuda koja nastaje na temelju procjene. Stoga je jedan od velikih izazova veća primjena rezultata istraživanja u svakodnevnoj industrijskoj primjeni.

S obzirom na sve navedeno u ovom radu, jasno je zbog čega su metrike i metode mjerenja i procjenjivanja važan dio softverskog inženjerstva, odnosno softverske industrije.

LITERATURA

[1] Vresk, A., Pihir, I., Tomičić Furjan, M.: Agilne vs tradicionalne metode za upravljanje IT projektima – studija slučaja, Varaždin, FOI, 2020.

[2] Čelar, S., Šeremet, Ž., Marušić, Ž., Turić, M. Using of Web Objects Method in Agile Web Software Projects, 21st Telecommunications Forum (TELFOR), proceedings of papers, Paunović, George (ur.). Beograd: IEEE, 2013. str. 873-876

[3] Arlow, J., Neustadt, I.: UML 2 and the Unified Process, 2nd Edition, Addison-Weslwy, Upper Saddle River NJ, USA, 2005. ISBN 0321321278

[4] Jones, C. (2008). Applied Software Measurement – Global Analysis Of Productivity And Quality, 3rd ed. New York, USA: McGraw-Hill, 2008.

[5] Beck, K., Andres, C.: Extreme Programming Explaind – Embrace Change, 2nd Edition, Addison-Wesley, Upper Saddle River NJ, USA, 2004. ISBN 0321278658

[6] Williams, L.: What agile Teams Think of Agile Principles, Communications of the ACM, 2012. DOI: 10.1145/2133806.2133823

[7] Williams, L.: Agile Software Development Metodologies and Practices, Advances in Computers, VOL. 80, 2010. ISSN: 0065-2458/DOI: 10.1016/S0065-2458(10)80001-4

[8] Doyle, M., Williams, L., Cohn, M., Rubin, K. S.: Agile Software Development in Practice, Springer, LNBIP 179, pp. 32-45, 2014.

[9] Lynn Cooke, J.: Agile Productivity Unleashed, IT Governance Publishing, UK, 2010. ISBN 978-1-84928-072-3

[10] Masood, Z., Hoda, R., Blincoe, K., „Real World Scrum A Grounded Theory of Variations in Practice“, IEEE Transactions on Software Engineering, vol. 48, no. 5, pp. 1579-1591, 1 May 2022, DOI: 10.1109/TSE.2020.3025317

[11] Ward, R., Chang, C. K., "Scrum, Sampling, and the 90 Percent Syndrome," *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1004-1013, DOI: 10.1109/COMPSAC51774.2021.00136

[12] Celar, S., Turic, M., Vickovic, L. Method for personal capability assessment in agile teams using personal points, 22nd Telecommunications Forum, Paunović, George (ur.). Beograd: IEEE, 2014. str. 1134-1137

[13] Popović, J., & Bojić, D. (2012). A comparative evaluation of effort estimation methods in the software life cycle. Computer Science and Information Systems, 9(1), 455–484. https://doi.org/10.2298/CSIS110316068P

[14] Minkiewicz, A. (2008). The Evolution of Software Size: A Search for Value, Software TechNews, Volume 11 (3), Data & Analysis Center for Software, 2008, pp. 18-22

[15] Albrecht. A.J., Gaffney. J., Jr. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Trans. Softw. Eng. Volume SE-9, No. 6, IEEE Press (Nov. 1983), pp. 639-648.

[16] Albrecht AJ. Measuring application development productivity. Proceedings of the joint SHARE/GUIDE and IBM Application Development Symposium. SHARE, Inc. and GUIDE Intl. Corp.: Chicagho IL, 1979; 83–92.

[17] Boehm, B. et al. (1999). COCOMO II model definition manual. Center for Software Engineering, 1999.

[18] ISO/IEC 14143-1:1998 (1998). Information technology – Software measurement – Functional Measurement – Part 1: Definition of Concepts. JTC1/SC 7, ISO/IEC, 1998.

[19] S. S. Ali, M. Shoaib Zafar and M. T. Saeed, "Effort Estimation Problems in Software Maintenance – A Survey," *2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 2020, pp. 1-9, doi: 10.1109/iCoMET48670.2020.9073823.

[20] Čelar, S.; Vicković, L.; Mudnić, E. Evolutionary Measurement-Estimation Method for Micro, Small And Medium-Sized Enterprises Based on Estimation Objects, Advances in Production Engineering & Management (APEM), **7** (2012), 2; pp. 81-92

[21] C. Gencel and O. Demirors, "Conceptual Differences Among Functional Size Measurement Methods," *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 2007, pp. 305-313, doi: 10.1109/ESEM.2007.43.

[22] ISO/IEC 20926:2009 Software and systems engineering — Software measurement — IFPUG functional size measurement method 2009 (rev. 2019), s Interneta: https://www.iso.org/standard/51717.html, zadnji pristup: 24.08.2022.

[23] Jones, C. (2008). A new business model for function point metrics. Capers Jones & Associates Llc, 2008.

[24] Ozkan, B. (2008). The Effect of Entity Generalization on Software Functional Sizing: A Case Study. Lecture Notes in Computer Science.

[25] Quesada-López, C., Martínez, A., Jenkins, M., Salas, L.C., Gómez, J.C. (2019). Automated Functional Size Measurement: A Multiple Case Study in the Industry. In: Franch, X., Männistö, T., Martínez-Fernández, S. (eds) Product-Focused Software Process Improvement. PROFES 2019. Lecture Notes in Computer Science(), vol 11915. Springer, Cham., DOI: 10.1007/978-3-030-35333-9_19

[26] Stephen H. Kan; Metrics and Models in Software Quality Engineering, 2nd edition, Addison-Wesley Longman Publishing Co., Inc. Boston, USA, ISBN: 0201729156, 2002

[27] Cohn, M. (2005). Agile estimating and planning. New York, USA: Prentice Hall, 2005.

[28] Sarro, F., Moussa, R., Petrozziello, A. Harman, M. Learning From Mistakes: Machine Learning Enhanced Human Expert Effort Estimates, IEEE Transactions on Software Engineering, Vol. 48, NO. 6, JUNE 2022, pp. 1868-1882

[29] Leung, H., Fan, Z.; Software cost estimation, Handbook of Software Engineering, Department of Computing The Hong Kong Polytechnic University, 2002

[30] Boehm B., Bradford C., Horowitz E., Westland C., Madachy R., Selby R.; Cost models for future software life cycle processes: COCOMO 2.0, Annals of Software Engineering, Online ISSN: 1573-7489, 1995, pp. 57-94

[31] Kitchenham, B., Mendes, E.; Why comparative effort prediction studies may be invalid, In Proceedings of the 5th International Conference on Predictor Models in Software Engineering, ACM, 2009, p. 4

[32] Meli, R., Abran, A., Ho, V. T., i Oligny, S. 2000. On the applicability of COSMIC-FFP for measuring software throughout its life cycle. *Proceedings of the Escom-Scope*, 2000.

[33] Boehm B., Abts C., Chulani S.; Software development cost estimation approaches – A survey, Annals of Software Engineering, Online ISSN: 1573-7489, 2000, pp. 177-205

[34] Bramble, M., Hihn, J., Hackney, J., Khorrami, M., Monson, E.; Handbook for Software Cost Estimation, Jet Propulsion Laboratory Pasadena, California, JPL D-26303, 2003

[35] Jørgensen, M.: A Review of Studies on Expert Estimation of Software Development Effort, 2004.

[36] Hamid Habib-agahi, Jairus Hihn: Cost Estimation of Software Intensive Projects: A Survey of Current Practices, 1991.

[37] Barbara A Kitchenham, Shari Lawrence Pfleeger i ostali: Preliminary guidelines for empirical research in software engineering, 2002.

[38] Lederer, A. L., Prasad, J. Nine: Management Guidelines for Better Cost Estimating, 1992

[39] CHAOS Report 2015: https://www.infoq.com/articles/standish-chaos-2015, s Interneta, zadnji pristup: 20.01.2019.

[40] Wysocki, K.R., McGary, R.: Effective Project Management, 3rd Edition, John Wiley & Sons, 2003.

[41] McConnell, S. (2006). Software Estimation: Demystifying the Black Art, WA, USA: Microsoft Press, 2006.

[42] Laranjeira L.A.: Software size estimation of object-oriented systems, 1990

[43] Patric Howard, Cone of uncertainty, s Interneta: https://appliedproductmanagement.wordpress.com/2016/11/15/cone-of-uncertainty/, zadnji pristup: 15.12.2018

[44] Goyal, S. Effective Software Effort Estimation using Heterogenous Stacked Ensemble, (2022) *SPICES 2022 – IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems*, pp. 584-588. DOI: 10.1109/SPICES52834.2022.9774231

[45] Kapur, R., Sodhi, B. OSS Effort Estimation Using Software Features Similarity and Developer Activity-Based Metrics (2022) *ACM Transactions on Software Engineering and Methodology*, 31 (2), art. no. 33, DOI: 10.1145/3485819

[46] Asres, M.W., Ardito, L., Patti, E. Computational Cost Analysis and Data-Driven Predictive Modeling of Cloud-based Online NILM Algorithm (2021) *IEEE Transactions on Cloud Computing*, DOI: 10.1109/TCC.2021.3051766

[47] Dragičević, S., Čelar, S., Turić, M. Bayesian network model for task effort estimation in agile software development, Journal of systems and software, 127 (2017), 109-119 doi: 10.1016/j.jss.2017.01.027

[48] Vaidhyanathan, K., Chandran, A., Muccini H., Roy, R. (2022) "Agile4MLS – Leveraging Agile Practices for Developing ML-enabled systems: An Industrial Experience," in *IEEE Software*, DOI: 10.1109/MS.2022.3195432

[49] Rashid, J., Kanwal, S., Nisar, M.W., Kim, J., Hussain, A. An Artificial Neural Network-Based Model for Effective Software Development Effort Estimation, (2023) Computer Systems Science and Engineering, 44 (2), pp. 1309-1324.

[50] Choetkiertikul M, Dam HK, Tran T, Pham T, Ghose A, Menzies T (2018) A deep learning model for estimating story points. IEEE T Softw Eng 45(7):637–656

[51] Kaushik, A., Kaur, P., Choudhary, N. et al. Stacking regularization in analogy-based software effort estimation. Soft Computing, 26, 1197–1216 (2022). https://doi.org/10.1007/s00500-021-06564-w

[52] Rankovic, D., Rankovic, N., Ivanovic, M., Lazic, L. Convergence rate of Artificial Neural Networks for estimation in software development projects, Information and Software Technology, Volume 138, October 2021, 106627, https://doi.org/10.1016/j.infsof.2021.106627

PRILOZI

Kazalo slika

Kazalo tablica

Popis oznaka i kratica

COCOMO II	*eng. Constructive Cost Model* II (funkcijski algoritamski model II)
COCOMO	*eng. Constructive Cost Model* (funkcijski algoritamski model)
COSMIC	*eng. COmmon Software Measurement International Consortium* (metoda bazirana na funkcijskim točkama)
DSDM	*eng. Dynamic systems development* (agilna metodologija)
EIF	*eng. External Interface File* (eksterni podatci za razmjenu)
FDD	*eng. Feature driven development* (agilna metodologija)
FiSMA	*eng. Finland Software Metric Association* (Finska udruga za softversku metriku)
FPA	*eng. Function point Analysis* (analiza funkcijskim točkama)
FSM	*eng. Functional Size Measurement* (mjerenje funkcionalnosti funkcijskim točkama)

IBM	*eng. International Business Machines Corporation* (američka tvrtka koja je jedan od pionira u razvoju računarstva i informacijske tehnologije)
IFPUG	*eng. International Function Point Users Group* (stručno udruženje)
ILF	*eng. Internal Logical Files* (interne logičke datoteke)
ISO	*eng. International Organization for Standardization* (međunarodna organizacija za normizaciju)
ISO/IEC 20926	standard za metodu mjerenja u softverskom inženjerstvu
ISO/IEC 20968	standard za metodu mjerenja u softverskom inženjerstvu
ISO/IEC 24570:2005	standard za metodu mjerenja u softverskom inženjerstvu
ISO/IEC 29881:2008	standard za metodu mjerenja u softverskom inženjerstvu
IT	*eng. Information technology* (Informacijska tehnologija)
KLOC	*eng. Thousands Line of Code* (tisuću linija koda)
LOC	*eng. Line of Code* (metrika preko broja linija koda)
MMRE	eng. *mean magnitude relative error – MMRE* (relativna pogreška srednje veličine)
NESMA	*eng. Netherlands Software Metric Association* (Nizozemska udruga za softversku metriku)
SLIM	softverski alat temeljen na Putnam modelu za procjenu troškova softvera i raspoređivanja radne snage
SP	*eng. Story Point* (mjerna jedinica za mjerenje veličine ukupne korisničke priče)
SW	*eng. Software* (računalni program)
UCP	*eng. Use Case Point* (mjerna jedinica za mjerenje/izražavanje funkcionalnosti i kompleksnosti softvera)

UML	*eng. Unified Modeling Language* (razvojni jezik modeliranja opće namjene u području softverskog inženjerstva)
XP	*eng. Extreme Programming* (metodologija Ekstremnog programiranja)

SAŽETAK I KLJUČNE RIJEČI

Sažetak

U radu su predstavljene metodologije razvoja softvera s naglaskom na agilni razvoj, te opisane one najzastupljenije poput Ekstremnog programiranja (XP) i Scrum-a. Obrađeno je mjerenje funkcionalnosti softvera, njegov početak, metode, problemi i odnos s procjenjivanjem. Također, obrađeno je i područje procjenjivanja gdje su opisane najznačajnije metode poput procjene po analogiji, procjene stručnjaka i sl. Izneseni su problemi i izazovi o (ne)uspješnosti softverskih projekata. Poseban naglasak stavljen je na primjenu strojnog učenja i umjetne inteligencije u procjenjivanju ali i na izazove procjenjivanja u agilnom razvoju.

Ključne riječi

Razvoj softvera, agilni razvoj, mjerenje u softverskom inženjerstvu, procjenjivanje u softverskom inženjerstvu