4 PROCESNI MODELI

Programski inženiring je uporaba discipliniranega, merljivega in sistematičnega pristopa v razvoj, zagon in vzdrževanje programske opreme. Temelj programskega inženiringa predstavlja proces (Pressman in Maxim 2014, 15). V informacijskih sistemih je definiran kot pot, ki jo moramo upoštevati, da pridemo do produkta (Rolland 1998, 3). Proces je zbirka aktivnosti, del in nalog, ki se izvajajo ob ustvarjanju produkta. (Pressman in Maxim 2014, 16). Procesi so kompleksne aktivnosti, kateri lahko vsebujejo tudi sebi podrejene procese. (Sommerville 2010, 28). Vse te aktivnosti bivajo v okvirjih ali modelih, ki definirajo njihove povezave s procesom in drugimi aktivnostmi (Pressman in Maxim 2014, 31). Kot tudi ostali intelektualni in kreativni procesi tudi ti slonijo na odločitvah posameznikov. Procesi so se razvili, da bi izkoristili sposobnosti ljudi v organizaciji in karakteristike sistema katerega razvijajo. (Sommerville 2010, 28). Procesi iste narave so razdeljeni v procesne modele (Rolland 1998, 8). Procesni model je poenostavljena predstavitev procesa razvoja. Vsak model predstavlja proces z določene perspektive in tako podaja le delček informacije o dejanskem procesu (Sommerville 2010, 29). Procesni model je bolj ali manj grobo pričakovanje poteka procesa, ki ga dejansko vidimo šele ob uporabi v praksi (Rolland 1998, 8). Pri uporabi se modeli med seboj ne izključujejo. V velikih sistemih se jih pogosto uporablja več skupaj (Sommerville 2010, 30). Procesni model je primarni pristop, ki projekt organizira v aktivnosti (Lethbridge in Laganiere 2005, 428). Obstajajo standardni modeli procesov, ki ne predstavljajo končnega procesa razvoja ampak služijo kot abstraktne forme s katerimi lahko opišemo različne pristope razvoja programske opreme. (Sommerville 2010, 29).

4.1 PRILOŽNOSTNI MODEL

Organizacije se velikokrat ne poslužujejo dobrim praksam razvoja programske opreme. Zaradi česar končajo pri uporabi modela, ki ga imenujemo priložnostni model. Pri tem pristopu, ki je viden na sliki 4.1, razvijalci konstantno modificirajo svoj produkt dokler ta ne ustreza njim ali naročniku (Lethbridge in Laganiere 2005, 428).

Slika 4.1 priložnostni model

  
vir: (Lethbridge in Laganiere 2005, 428)

Slabost tega pristopa je, da pred začetkom implementacije ne predvideva pomembnosti zahtev in načrtovanja. Načrtovanje v tem modelu je ad hoc aktivnost. Ker ne vsebuje podrobnih načrtov ni jasnih ciljev, posledično ni jasne slike ali gre razvoj v pravo smer ali ne. Pristop eksplicitno ne predvideva sistematičnega testiranja ali drugih načinov zagotavljanja kakovosti kar pomeni, da lahko končen produkt vsebuje pomanjkljivosti. (Lethbridge in Laganiere 2005, 428).

4.2 KASKADNI MODEL

Pomenljiv napredek priložnostnemu modelu predstavlja kaskadni model (waterfall model) (Lethbridge in Laganiere 2005, 428). Kaskadni model, imenovan tudi klasični življenjski cikel, narekuje sekvenčni, sistematski pristop k razvoju programske opreme. Začne se s specifikacijo potreb naročnika, nato se nadaljuje v načrtovanje, modeliranje, konstrukcijo in zagon, ki kulminira v nadaljnji podpori končanega projekta (gl. Sliko 4.2) (Pressman in Maxim 2014, 42).

Slika 4.2. Kaskadni model po Pressman in Maxim

  
vir: (Pressman in Maxim 2014, 42)

Model predstavlja klasični pogled na inženiring programske opreme z upoštevanjem pomembnosti potreb, načrtovanja in zagotavljanja kakovosti. (Lethbridge in Laganiere 2005, 428). Model je primer načrtno-usmerjenih procesov, ker je v praksi vsako aktivnost potrebno načrtovati. (Sommerville 2010, 31). Sommerville (2010) bolj podrobno opiše posamezne korake modela. Osnovni koraki povsem opišejo temelje aktivnosti modela (gl. Sliko 4.3). Sistemske zahteve in cilji so vzpostavljeni preko komunikacije z uporabniki ali naročniki. Ti oblikujejo sistemsko specifikacijo. Proces načrtovanja vzpostavi sistemsko arhitekturo na podlagi zahtev po programski ali strojni opremi. Implementacija in testiranje enot služi za verifikacijo delovanja posameznih fragmentov sistema na podlagi sistemske specifikacije. Naslednji korak prestavlja testiranje in verifikacijo celotnega sistema in kasnejšo implementacijo ob ustreznosti delovanja na podlagi sistemske specifikacije (Sommerville 2010, 31).

Slika 4.3 Kaskadni model po Sommerville  
vir: (Sommerville 2010, 31)

Načeloma je rezultat vsakega koraka potrditev enega ali več dokumentov. (Sommerville 2010, 32). Validacija in zagotavljanje kakovosti pri vsakem koraku omogoča, da se naslednji korak prične na trdnih temeljih (Lethbridge in Laganiere 2005, 429). Čeprav se praviloma naslednji korak ne more pričeti dokler prejšnji ni zaključen se koraki v praksi deloma prekrivajo in prenašajo informacije eden do drugega. Dokumenti, ki so že bili potrjeni se morajo spremeniti, da ustrezajo spremembam v korakih. Zaradi ponovnega pregledovanja dokumentacije lahko te iteracije predstavljajo dodatne stroške in občutno več dela. Težave so velikokrat zamrznjene, prestavljene za kasnejšo obravnavo, prezrte ali zaobite. Za kakršenkoli poseg v sistem, ki je že v delovanju, bo razvoj najverjetneje potreboval ponoviti pretekle korake procesa (Sommerville 2010, 32). Naročnik je navadno prisoten pri podajanju zahtev, potem večinoma ponikne v fazi analiz, načrtovanja ali kodiranja. Pojavi se zopet pri testiranju in predaji produkta (Palmquist in dr. 2013, 5). Kasnejšim zahtevam naročnika je zato težko ugoditi (Sommerville 2010, 32). Ta okorelost je še posebej moteča ob dejstvu, da se zahteve naročnikov konstantno spreminjajo. Ob zaključku procesa model predvideva le vzdrževanje. Nadaljnji razvoj se razume kot manjši poseg, kjer ni potreb po ponovnem načrtovanju. Na žalost se velikokrat izkaže da temu ni tako, saj se sistem konstantno spreminja (Lethbridge in Laganiere 2005, 429).

Zaradi dokumentacije, ki je proizvedena na koncu vsakega koraka je ta model transparenten (Sommerville 2010, 32). Stroga kontrola nad ustreznostjo dokumentacije ima pozitiven vpliv na kvaliteto, zanesljivost in vzdržljivost programske opreme (CMS 2005, 2). Napredek razvoja je merljiv (CMS 2005, 2) in lažje primerljiv s časovnim načrtom (Sommerville 2010, 32). Model je idealen za manj izkušene projektne skupine in manj izkušeno vodstvo (CMS 2005, 2). V primeru dobro definiranih zahtev naročnika, nominalni verjetnosti sprememb (Sommerville 2010, 32), ob pričakovanju večje fluktuacije zaposlenih, večjem, časovno fleksibilnem projektu z daljšim razvojnim ciklom in potrebi po ohranitvi sredstev je ta model primeren za uporabo (CMS 2005, 2).

Slika 4.4 Kaskadni model po Lethbride in Laganiere  
  
vir: (Lethbridge in Laganiere 2005, 429)

V literaturi se pojavljajo različne slike, ki ta model opisujejo z različnimi ali več koraki. Za primer so slike 4.2, 4.3 (gl. str. X) in sliko 4.4 (gl. str. X). Kot je razvidno vse slike ustrezajo klasičnemu pogledu inženiringa programske opreme. Kot tak se mora začeti s potrebami, nadaljevati v načrtovanju in končati z zagotavljanjem kakovosti. Takšen pogled ustreza kaskadnemu modelu. Proces specifikacije in pridobivanja informacij je vedno pred načrtovanjem, implementacijo in zagotavljanjem kakovosti. Načrtovanje in implementacija se vedno odvijata po pridobljenih informacijah in zagotavljanje kakovosti je v vseh primerih kaskadnega modela na zadnjem mestu.

4.3 V-MODEL

Variacija kaskadnega modela se imenuje V-model (Pressman in Maxim 2014, 43). Imenovan tudi kot variacijski in validacijski model (Tutorialspoint 2017). Verifikacija in validacija (V&V) sta namenjeni za potrditev ustreznosti programske opreme na podlagi specifikacije in pričakovanj naročnika. (Sommerville 2010, 41). Ta model prikazuje odnose med aktivnostmi zagotavljanja kvalitete in komunikacije (gl. sl. 4.5). S pomikanjem razvijalcev po levi strani V-modela se osnovni problem preoblikuje v vedno bolj podrobno in tehnično reprezentativno rešitev. Ko je del kode generiran se ekipa pomika po drugi strani V-modela navzgor in opravi serijo preizkusov. V osnovi ni razlike med kaskadnim in v-modelom. Slednji le pripomore k vizualizaciji poteka verifikacije in validacije posameznih korakov v kaskadnem modelu (Pressman in Maxim 2014, 43).

Slika 4.5 V model po Pressman in Maxim  
  
vir: (Pressman in Maxim 2014, 43)

Model predvideva, da se zahteve, njihove prioritete in red ne bodo spreminjale, kar je moteče če upoštevamo, da vsak korak skriva potencialne napake. Prvi testi so narejeni daleč v razvoju, kar je pozno in stane veliko. (Palmquist in dr. 2013, 9).

4.4 INKREMENTALNI MODEL

V modelu se združujejo elementi linearnega in paralelnega procesnega toka. V časovnem toku se odvijajo linearne sekvence na različnih stopnjah. Vsaka sekvenca predstavlja del sistema programa v razvoju. Posamezno sekvenco imenuje inkrement (Pressman in Maxim 2014, 44). Iz slike 4.6 je razvidno, da inkrementalni model vsebuje sledi klasične paradigme inženiringa programske opreme, saj posamezni inkrement upošteva sosledje potreb, načrtovanja in zagotavljanja kakovosti. Primarni namen tega modela je zmanjšati tveganje razvoja s fragmentacijo projekta v manjše celote. (CMS 2005, 5).

Slika 4.6 inkrementalni model po Pressman in Maxim  
vir: (Pressman in Maxim 2014, 44)

Prvi inkrement je navadno jedro produkta (Pressman in Maxim 2014, 44). Vsebuje najpomembnejše funkcionalnosti (Sommerville 2010, 33) Ta je uporabljen za naročnikovo evaluacijo na podlagi katere se oblikuje načrt za nov inkrement. Proces se ponavlja z izdajanjem vedno novejšega in popolnejšega inkrementa do končnega produkta. Ker se produkt razvija po stopnjah je ceneje in lažje uveljavljati spremembe (Pressman in Maxim 2014, 44). Investitor lahko ob vsaki stopnji pridobi stvarno predstavo statusa projekta skozi razvoj. (CMS 2005, 5).

Tak pristop razvoja je danes najpogostejši. Model velja za temelj agilnim pristopom, kljub temu pa je lahko načrtno-usmerjen, agilen ali agregat obeh. V načrtno-usmerjenih so inkrementi definirani vnaprej. Ob asimilaciji agilne umeritve pa so zgodnji inkrementi izdelani, prihodnji pa so odvisni od prioritet naročnika (Sommerville 2010, 34). Postopne implementacije omogočajo spremljanje vpliva vgrajenih sprememb, zajezitev težav in prilagoditev preden bi lahko te ogrožale delovanje organizacije. Model potrebuje le zmeren poseg vodenja, ki se izvaja na podlagi dokumentacije in formalnih kritik po končanih večjih razvojnih mejnikih (CMS 2005, 5).

Uporaba tega modela je težavna pri velikih, kompleksnih in dolgoročnih projektih kjer različne delovne skupine posamično razvijajo dele programske opreme. (Sommerville 2010, 34). Pred začetkom novega inkrementa so tehnične zahteve splošnega, končnega sistema pogosto spregledane (CMS 2005, 5).

4.5 EVOLUCIJSKI PROCESNI MODELI

Med razvojem se zahteve pogosto spreminjajo kar onemogoča linearnost procesov. Tesni razvojni cikli onemogočajo dokončanje zapletenih programskih paketov. Zavoljo poslovnih in konkurenčnih pritiskov pa mora razvoj vseeno izdati omejeno različico paketa. Kadar imamo produkt, ki se konstantno razvija in spreminja, ko so temeljne zahteve programske opreme dobro poznane, razširitve slednjih pa še v načrtovanju so najprimernejši evolucijski procesni modeli. Ti modeli so ponavljajoči se, iterativni. Sem spadajo prototipiranje, spiralni in sočasni procesni model. (Pressman in Maxim 2014, 45).

4.5.1 PROTOTIPIRANJE

Čeprav lahko ta model uporabljamo samostojno pa se ga v praksi največkrat uporablja znotraj konteksta kateregakoli drugega procesnega modela.

Sekvenca korakov predstavlja faze procesa. Ta pristop je v osnovi sestavljen iz prepleta specifikacije, razvoja in validacije, ki sestavljajo sosledje različic projekta. Tak pristop je inkrementalen pri čemer vsaka naslednja različica doda funkcionalnost prejšnji. (Sommerville 2010, 30).

Procesni model ni enostaven linearen model ampak vključuje potrebo po povratnih informacijah od koraka do koraka (Sommerville 2010, 31).

Parts of the system which are difficult to

specify in advance, such as the user interface, should always be developed using an

incremental approach.(Sommerville 2010, 30)

Cleanroom software engineering

An example of a formal development process, originally developed by IBM, is the Cleanroom process. In the

Cleanroom process each software increment is formally specified and this specification is transformed into an

implementation. Software correctness is demonstrated using a formal approach. There is no unit testing for

defects in the process and the system testing is focused on assessing the system’s reliability.

The objective of the Cleanroom process is zero-defects software so that delivered systems have a high level

of reliability.

<http://www.SoftwareEngineering-9.com/Web/Cleanroom/>

(Sommerville 2010, 32).

An important variant of the waterfall model is **formal system development**, where

a mathematical model of a system specification is created. This model is then

refined, using mathematical transformations that preserve its consistency, into executable

code. Based on the assumption that your mathematical transformations are

correct, you can therefore make a strong argument that a program generated in this

way is consistent with its specification.

Formal development processes, such as that based on the B method (Schneider,

2001; Wordsworth, 1996) are particularly suited to the development of systems that

have stringent safety, reliability, or security requirements. The formal approach simplifies

the production of a safety or security case. This demonstrates to customers or

regulators that the system actually meets its safety or security requirements.

Processes based on formal transformations are generally only used in the development

of safety-critical or security-critical systems. They require specialized

expertise. For the majority of systems this process does not offer significant costbenefits

over other approaches to system development. (Sommerville 2010, 32).

The waterfall model forms the foundation of many software development

methodologies in use today. However, it has some limitations and, if followed

too strictly, can lead to the following types of problems: (Lethbridge in Laganiere 2005, 429)

V uvod

Ne obstaja idealen proces in večina organizacij je razvila svoje procese razvoja (Sommerville 2010, 28). Čeprav ne obstaja idealen proces, velja, da v veliko organizacijah obstaja prostor za izboljšave. Procesi lahko uporabljajo zastarele tehnike ali pa ne izkoriščajo najnovejših in najboljših praks. Veliko organizacij še vedno ne uporablja najnovejših pristopov pri razvoju lastne programske opreme. Proces razvoja je lahko izboljšan s standardizacijo procesov. To vodi v izboljšano komunikacijo, manjšo porabo časa pri uvajanju in zmanjšanje stroškov avtomatiziranega vodenja. (Sommerville 2010, 29).

Pressman (2014) deli modele glede na njihovo strukturo in namembnost na perspektivne, specializirane, enotne, osebne in ekipne.

Navadno je najdlje trajajoča aktivnost vzdrževanje programske opreme (Sommerville 2010, 31).

Življenjski cikel razvoja programske opreme predstavlja organizacijsko shemo procesa razvoja (Glass 2002).

Proces razvoja programske opreme je skupek aktivnosti, ki vodijo v produkcijo programske opreme. (Pressman in Maxim 2014, 15).

Inženiring procesa razvoja programske opreme omogoča racionalen in časovno sprejemljiv razvoj programske opreme (Pressman in Maxim 2014, 15).

Standardni procesni modeli definirajo pet osnovnih aktivnosti v modelu: komunikacija, načrtovanje, modeliranje, izdelava in uvajanje (Pressman in Maxim 2014, 31).