5 INŽENIRING METOD

Inženiring metod (IM) je disciplina, ki se ukvarja z razvojem metod, tehnik in orodij za razvoj informacijskih sistemov (Engels in Sauer 2010, 412). Slika 5.1 prikazuje model splošnega življenjskega cikla IM. Po identifikaciji domene diskurza (razvoj programske opreme) se prične analiza potreb, ki predstavlja prvi korak. Sledi ji več-stopenjski razvojni proces. Po razvoju se metodo namesti v uporabo in preveri. Preverjanje privede do novih ugotovitev ali potreb, katere vodijo v zagon ponovnega evolucijskega cikla metode (Engels in Sauer 2010, 422).

Slik 5.1 življenjski cikel IM



Vir: (Engels in Sauer 2010, 422)

Cilj inženiringa ni le definiranje procesa in vseh njegovih aktivnosti, temveč se osredotoča tudi na produkte, naloge, organizacijske vloge, orodja, tehnike in medsebojne relacije teh konceptov, zato takšne procese označuje za metode inženiringa programske opreme (MIPO) (Engels in Sauer 2010, 411-12). MIPO so sistematični postopki ali tehnike izvajanja opravil z namenom doseganja določenih ciljev ali definiranjem zbirke artefaktov programske opreme (Engels in Sauer 2010, 414). MIPO koordinirajo in dokumentirajo razvojne procese, aktivnosti in artefakte. Strokovnjaki jih pogosto imenujejo procesni modeli (Engels in Sauer 2010, 415).

Procesni model ustreza načinu dela, ki ga predpisuje metodologija. Model lahko razumemo tudi kot koncept načrta. Za oblikovanje takšnih načrtov je potrebna uporaba abstrakcije, ki jo predstavlja meta-model (sl. 5.1). Procesni model (Plast 2) je v praksi tako instanca meta-modela (Plast 3) (Rolland in Ufr 1994, 3). Pri procesnem modeliranju je velikokrat spregledano dejstvo, da je procesni model abstraktna predstavitev procesa in kot taka predstavlja proces iz drugačne perspektive, ki lahko definira potek dela kateri še ni v praksi (Peters 2008, 105).

Slika 5.1 nivoji abstrakcije pri procesnem modeliranju

  
vir: (Rolland in Ufr 1994, 3)

MIPO so v večini primerov sestavljene iz več delov kateri delijo proces razvoja na obvladujoče entitete (Sunyaev, Hansen, in Krcmar 2009, 646). Entitete v stroki predstavljajo kose, fragmente ali komponente. Poimenovanja so odvisna od pristopov k strukturiranju metod in granulacije posameznih entitet (Sunyaev, Hansen, in Krcmar 2009, 648). Harmsen, Brinkkemper in Oei označijo fragmente za osnovne gradnike metod. Na podlagi tega se razvijeta produktna in procesna dimenzija fragmentov metod (Sunyaev, Hansen, in Krcmar 2009, 647). Produktna dimenzija definira artefakte, ki se morajo izdelati in njihovo medsebojno povezanost (Engels in Sauer 2010, 413). Artefakti (produktni fragmenti) so ciljno orientirani in jih predstavljajo produkti, dokumenti, modeli, diagrami in drugo. Procesna dimenzija pa mora definirati postopke, ki so potrebni za izdelavo artefaktov in njihove izvajalce (Engels in Sauer 2010, 413). Ti postopki (procesni fragmenti) so procesno orientirani in jih predstavljajo faze, naloge in aktivnosti (Sunyaev, Hansen, in Krcmar 2009, 647).

Ralyté in Rolland nadgradita pristop fragmentacije metod z uporabo kosov. Obstoječe procese in produktne fragmente združita v kose metod, ki zagotavljajo tesno povezavo procesnega in produktnega vidika (Sunyaev, Hansen, in Krcmar 2009, 648). Karlson in Wistrand pa delita metode na komponente. Komponento smatrata za najmanjšo smiselno entiteto metod katera sestoji iz procesa, notacije in koncepta (Sunyaev, Hansen, in Krcmar 2009, 648).

5.1 PROCESI

Procesi razvoja programske opreme so definirani z IEEE[[1]](#footnote-1) standardom 610.12, ki navaja: ''Postopek, s katerim se potrebe uporabnikov prevedejo v programsko opremo'' („IEEE 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology“ 2017). Definirajo pravila in priporočila za razvoj informacijskih sistemov in podajajo informacijo v kakšnem zaporedju izvajati ukrepe (Sunyaev, Hansen, in Krcmar 2009, 648). So časovno razporejene zbirke aktivnosti, ki so usmerjene v cilj ali izid (Peters 2008, 103). Podrobneje jih lahko opišemo kot zbirko deloma decidiranih korakov s podskupino medsebojno povezanih artefaktov, fizičnih in virtualnih virov, organizacijskih struktur in omejitev z namenom proizvajanja in vzdrževanja programske opreme (García-Borgoñón in dr. 2014, 104). Proces je predstavljen s procesnim modelom. Razumemo ga kot sestav delovnih elementov (aktivnosti, naloge, faze) in toka dela, ki opredeljuje njihovo časovno sosledje (Engels in Sauer 2010, 425).

Proces zajema aktivnosti, ki predstavljajo translacije uporabniških potreb v specifikacijo, transformacijo specifikacije v načrt, implementacijo načrta v kodo, testiranje kode in integracijo. Aktivnosti se lahko prekrivajo ali si iterativno sledijo. Aktivnosti predstavljajo hierarhične kompozite katere skupaj tvorijo potek dela inženirske metode programske opreme (Engels in Sauer 2010, 415).

S procesi sta povezana tudi dva termina, ki tvorita časovno perspektivo procesa razvoja programske opreme: cikel razvoja programske opreme in življenjski cikel razvoja programske opreme. Osnovna oblika vseh življenjskih ciklov je prikaz napredka od problema do rešitve (Peters 2008, 108). Potrebno je potrebno identificirati, načrtovati, organizirati in spremljati vse aktivnosti, ki so potrebne za izgradnjo novega sistema (Satzinger, Jackson, in Burd 2011, 5). Življenjski cikel razvoja identificira vse aktivnosti, ki so potrebne za izgradnjo, zagon in vzdrževanje informacijskih sistemov. Čeprav je mnogo variacij procesov znotraj življenjskih ciklov veljajo naslednje za osnovne;

1. Identifikacija problema ali potreb in odobritev za nadaljevanje,
2. načrtovanje in spremljanje projekta,
3. odkrivanje in razumevanje podrobnosti problema ali potreb,
4. oblikovanje sistemskih komponent, ki rešijo problem ali zadovoljijo potrebam,
5. izgradnja, testiranje in integracija sistemskih komponent,
6. testiranje sistema in prenos v delovno okolje (Satzinger, Jackson, in Burd 2011, 6).

5.2 NOTACIJE

Notacije predstavljajo semantična, sintaktična in simbolna pravila za dokumentiranje. (Sunyaev, Hansen, in Krcmar 2009, 648). Za definiranje procesa je potrebna globoka metodologija skupaj z močnim in ekspresivnim jezikom procesnega modeliranja (Jablonski 2010, 393). Jeziki modeliranja so bili izdelani na podlagi programerskih jezikov, Petrijevih mrež in jezikov, ki temeljijo na pravilih (García-Borgoñón in dr. 2014, 104). Uporabnost jezikov modeliranja se razlikuje glede na njihovo paradigmo. Nekateri so bolj primerni za nadzorovanje procesa, objektno-orientirani za nadzor funkcionalnosti in kode, podatkovni pa za nadzor in definiranje podatkovnih baz (Sutton 2011, 295). Procese lahko predstavimo na podoben način kot programsko opremo. Pri izgradnji se pogosto uporablja termin programiranje procesov, pri čemer beseda programiranje implicira obsežnost programskega inženiringa, ne le kodiranja. Čeprav strokovnjaki zagovarjajo algoritemski zapis procesov je potrebno upoštevati dejstvo, da razvoj programske opreme zajema ustvarjalne vidike, ki so inherentno ne algoritmični. Procesi so deležni posebnih, nepričakovanih iregularnosti, ki onemogočajo slednje algoritemskim predpisom (Sutton 2011, 293).

5.3 KONCEPTI

Vsem MIPO so skupni aspekti, kateri nakazujejo njihovo zgradbo:

* Delovni potek razvojnega procesa,
* aktivnosti, ki se morajo izvesti,
* definicije delovnih produktov ali njihovih fragmentov,
* kriterij zaključka delovnih produktov,
* sposobnosti in odgovornosti akterjev in
* standarde, usmeritve, tehnike in orodja (Engels in Sauer 2010, 417).

Avtorji znotraj teh aspektov uporabljajo različne konstrukte, ki jih inženirji metod imenujejo koncepti (gl. PRILOGA A) (Engels in Sauer 2010, 416).

Koncepti so kategorije, ki so vključene v procese in notacije. S koncepti opisujemo področja problema ali samo opravilo (Sunyaev, Hansen, in Krcmar 2009, 648). V praksi se srečamo z različnimi definicijami konceptov, pogosteje se uporabljajo: vloge (ljudje), procesi in aktivnosti (naloge), delavni produkti (artefakti), orodja (programska orodja in drugi pripomočki). Strokovnjaki se strinjajo, da mora programski inženiring zajemati vsaj tri koncepte: produkte, procese in akterje (Engels in Sauer 2010, 415).

6 METODE INŽENIRINGA

Za izgradnjo optimalnega procesa potrebujemo pristop, ki najbolje ustreza naši domeni uporabe. Inženiring metod nam podaja različne pristope k inženiringu procesov (Engels in Sauer 2010, 412). Kljub veliki izbiri pristopov je razbrati, da je v domeni razvoja video iger primerna uporaba agilne filozofije. Verjetnost za to obstaja, ker se produkcijski proces iger konstantno razvija. Tradicionalni procesi (derivati manufaktur) enostavno ne morejo dohajati hitrosti sprememb razvoja iger (Unger in Novak 2011, 180).

Izbiro metode nam nakazujejo igre in nameni uporabe. Igre se osredotočajo na igranje (Esposito 2005, 3). Igranje predstavlja element, ki emocionalno pritegne igralce. Je nekakšno stanje, ki je podobno zabavi (Fullerton 2008, 91). Posledično se igre smatrajo za najbolj ekstremne hedonične informacijske sisteme, pa vendar se pri utilitarnih namenih uporabe uporabna motivacija premakne iz zabave v uporabnost (Hamari in Keronen 2017, 136). Igre lahko vodijo tako intrinzične kot ekstrinzične motivacije (Novak 2012, 198). Številne študije kažejo pozitivne učinke učnih iger na področju motivacije do učenja in učnega izkoristka. Motivacija in pozitivne emocije sta pomembna faktorja pri učenju (Imlig-Iten in Petko 2014, 151), katerih uravnoteženost igre predstavlja efektivnost, ki je pridobljena s konsistenco zahtevnosti in zabave pri igranju (Novak 2012, 202). To predstavlja zabavno izkušnjo, ki je najpomembnejše merilo uspeha iger (Cooper in Scacchi 2015, 12). V razvoju iger se uporablja termin uporabniška izkušnja[[2]](#footnote-2), ki jo z iterativnim procesom upoštevanja uporabniških povratnih informacij oblikovalci izboljšujejo tekom procesa razvoja (Fullerton 2008, 2).

S povratnimi informacijami je povezana tudi pozornost na defekte. Zavoljo tega osrednja aktivnost razvoja video iger navadno postane konstantno testiranje[[3]](#footnote-3) igranja (Cooper in Scacchi 2015, 10). To aktivnost predstavlja testni protokol in velja za iterativni proces med načrtovanjem, razvojem in testiranjem (Sylvester 2013, 295). Zato se razvoji iger nagibajo od tradicionalnih življenjskih ciklov razvoja k bolj inkrementalnim, kateri na podlagi uporabniških povratnih informacij izpopolnjujejo ali izboljšujejo verzije programa (Cooper in Scacchi 2015, 10).

Po podatkih spletne ankete[[4]](#footnote-4) v Avstrijski industriji iger 23% podjetij razvija igre z ad-hoc pristopi, 77% pa jih uporablja Scrum ali XP. Verjetno je k rezultatu botrovala velikost samih podjetij, saj ima 85% podjetij vsaj 4 zaposlene medtem, ko ima le 15% podjetij 15 ali več zaposlenih. Ne glede na rezultate so vsa podjetja nakazala uporabo nekakšnih fleksibilnih, sekvenčnih ali agilnih pristopov (Musil in dr. 2010, 5).

Raziskava, ki so jo naredili O'Hagan in kolegi je pokazala, da se pri razvoju iger uporablja 47% agilnih in 53% hibridnih procesov. Izmed 404 študij so identificirali 23 procesnih modelov med katerimi so bili vidnejši: XP, Scrum, Kanban, rapidni, inkrementalni in komponentni razvojni modeli. Medtem, ko so prvi trije našteti strogo agili, vsi bazirajo na iteracijah. Vsi modeli raziskave so se razlikovali le po številu iteracij, katerih število je bilo večje pri agilnih in manjše pri hibridnih procesih (O’Hagan, Coleman, in O’Connor 2014, 187).

Prav tako agilne pristope podpira Fullerton, saj meni, da je Scrum primeren za reševanje zapletenih problemov oblikovanja iger (Fullerton 2008, 369).

Barbosa nadgradi Scrum z XP in ustvari Game-Scrum znotraj katerega se značilnosti XP-ja ukvarjajo z inženiringom programske opreme medtem, ko značilnosti Scrum-a skrbijo za projektno vodenje (Barbosa 2017, 293).

Ob uporabi agilnih razvojnih procesov delujejo bolje tudi mobilne igre. Razvijalci označujejo te metode za metode inkrementalnih in iterativnih komponent (Unger in Novak 2011, 178). Dejstvo je, da razvoj iger sloni na iteracijah (Novak 2012, 366) in kot smo spoznali tudi inkrementih. Iterativen in inkrementalen razvoj predstavljata temeljne principe agilnega modeliranja (Ambler 2002, 44).

Primerno metodo uporabe nam nakazuje tudi praksa uporabe procesnih modelov, ki evidentno sporoča uporabo agilnih pristopov in njim podobnih hibridov, ki slonijo na iteracijah. Spoznanja vodijo v prepričanje, da bi bila izbira metode agilnega modeliranja za izgradnjo procesnega modela upravičena.

Uporaba agilne metodologije kot samostojne metode ne bi zadostovala za izgradnjo sistematičnega procesa, saj potrebujemo proces v uporabi formalizirati. To pomeni, da je potrebno imeti proces dokumentiran, katerega je dobro frekvenčno revidirati (Rucker 2002, 32). Agilna metodologija je bolj produkcijska filozofija kot trdna zbirka pravil (Unger in Novak 2011, 181). Predstavlja skupek najboljših praks, ki slonjo na principih in vrednotah predstavljenih v Agilnem Manifestu[[5]](#footnote-5). Agilno modeliranje ne predpisuje postopkov za izgradnjo določenega procesa ampak spodbuja konstantno in efektivno modeliranje skozi nasvete in dobre prakse (Ambler 2002, 8).

Zato bomo uporabili dodaten pristop k modeliranju z namenom podpore dokumentaciji in sistematizaciji procesa. Obetaven pristop naproti sistematičnemu in strukturiranemu razvoju programske opreme je uporaba tehnik meta-modeliranja za izgradnjo procesnih modelov (Engels in Sauer 2010, 419).

6.1 AGILNO MODELIRANJE

Bistvo agilnega modeliranja (AM) leži v njegovih praksah. Prakse AM so organizirane v štiri kategorije: iterativno in inkrementalno modeliranje, skupinsko delo, enostavnost in validacija (Ambler 2002, 44–45).

Iterativno in inkrementalno kategorijo definirajo prakse:

* apliciranja pravih artefaktov,
* izdelave več vzporednih modelov,
* iteracije do naslednjega artefakta,
* izvajanja manjših inkrementov (Ambler 2002, 45).

Artefakte predstavljajo UML grafikon stanja, izvorna koda, diagram poteka podatkov, primeri uporabe in drugi (PRILOGA B). Vsak artefakt ima svoje prednosti pri uporabi v določenih situacijah (Ambler 2002, 45). V nekaterih primerih je bolj efektivno uporabiti diagram kot pa napisati 1024 vrstic kode. Pri modeliranju je pomembno razumevanje kdaj je smotrno uporabiti določen artefakt in kdaj ne (Ambler 2002, 46). Vzporedno modeliranje omogoča simultani zajem informacij več različnih artefaktov (Ambler 2002, 48). V primeru, da informacije postanejo neprimerne za določen artefakt se izvede iteracija do naslednjega artefakta. S tem se informacije prenesejo na drug artefakt in posledično omogočijo napredovanje v procesu. Proces se izvaja inkrementalno, kar predstavlja temelj agilnega modeliranja. Agilno modeliranje si prizadeva k fragmentaciji kompleksnejših nalog v manjše obvladljive entitete, ki se izvajajo v krajših intervalih. Daljši intervali se lahko zgodijo vendar so zgolj izjeme. Daljše kot je obdobje brez povratne informacije večja je nevarnost, da je bilo opravljeno delo zaman (Ambler 2002, 51). Daljša iteracija ne pomeni, da je bilo narejeno več dela. Agilni procesi se bolj fokusirajo na efektivne ure razvoja in ne na njihovo količino (Unger in Novak 2011, 180).

Skupinsko delo definirajo prakse:

* skupinskega modeliranja,
* aktivna participacija vlagateljev,
* kolektivno lastništvo,
* javni prikaz modelov (Ambler 2002, 44–45).

Skupinsko modeliranje omogoča boljše razumevanje idej in ustvarjanje skupne vizije projekta. Prav tako pripomore k izboljšani komunikaciji, izgradnji skupnega besednjaka in povečanju možnosti izvajanja kvalitetnega dela. Z izboljšano komunikacijo poda tudi priložnost prenosa znanja med zaposlenimi (Ambler 2002, 52). Navadno je za uspeh projekta potrebna večja stopnja vključenosti vlagateljev. Aktivna participacija tako spodbuja prisotnost uporabnikov ali vlagateljev na lokaciji z namenom izmenjave informacij glede zahtev in pričakovanj projekta (Ambler 2002, 53). Za kolektivno lastništvo velja, da lahko vsakdo sodeluje pri izgradnji določenega modela, kar predstavlja priložnost večje identifikacije napak (Ambler 2002, 54). Prav v ta namen se uporabljajo table za javni prikaz modelov, kar spodbuja odprto komunikacijo med zaposlenimi (Ambler 2002, 55).

Enostavnost definirajo prakse:

* ustvarjanja preproste vsebine,
* enostavnega prikaza modela,
* uporaba enostavnih orodij (Ambler 2002, 45).

Vsebino je priporočljivo poenostaviti do mere, da še vedno zadovoljuje projektu. To pomeni, da model še vedno zadovoljuje potrebe in ima svoj namen. Poleg tega model ne vsebuje podvojenih informacij in je sestavljen iz karseda nizkega števila elementov, kar vodi v enostaven prikaz (Ambler 2002, 56–57). Priporoča se izogibanje: križanim tranzicijam, zavitim tranzicijam, diagonalnim tranzicijam, različnim velikostim balonov, velikemu številu balonov (ne več kot 7 +/−2) in nepotrebnim podrobnostim (Ambler 2002, 57). Večina modelov lahko narišemo na tablo, kar je potrditev, da je fizična tabla najboljše orodje modeliranja. Uporaba enostavnih orodij deluje, saj je večina diagramov za enkratno uporabo (Ambler 2002, 58).

Validacijo definirajo prakse:

* upoštevanja preverljivosti,
* dokazovanja s kodo (Ambler 2002, 45).

Moderni procesi vključujejo aktivnosti testiranja in zagotavljanje kvalitete čez celoten življenjski cikel. Nekateri modeli celo priporočajo definiranje testnih konceptov pred izgradnjo sistema. Če ne moremo testirati sistema je bolje, da ga ne gradimo (Ambler 2002, 58). K temu stremi praksa upoštevanja preverljivosti. Ob zgrajenem modelu pa je potrebno preveriti ali je mogoča implementacija poslovnega pravila v model, torej ali ga bo mogoče izvajati. To lahko preverimo tudi z uporabo kode. V praksi se priporoča izvajati cikel modeliranja, kodiranja in testiranja (Ambler 2002, 59).

6.2 META MODELIRANJE

V literaturi je definiranih več pristopov meta-modeliranja različnih avtorjev. Med njimi sta dva standardizirana, ki uporabljata meta-modele za definiranje MIPO: ISO 24744:2007[[6]](#footnote-6) in SPEM[[7]](#footnote-7). Meta-model se osredotoča na prikaz vsebine metode in ima številne prednosti:

* Zagotavlja formalne temelje za specifikacijo MIPO,
* zgrajeni procesi so primerljivi na podlagi uporabe okvirja meta modela,
* formalizacija zagotavlja natančne temelje razvojnim orodjem,
* zagotavlja analiziranje konsistence in skladnosti,
* zagotavlja formalno podlago za modifikacije (Engels in Sauer 2010, 412–13).

Konzorcij OMG[[8]](#footnote-8) je v standardu MOF[[9]](#footnote-9) definiral štiri-plastno referenčno arhitekturo meta-modela na podlagi pogosto uporabljenih konceptov (Engels in Sauer 2010, 419). Kot je razvidno iz slike 6.2 se plasti delijo na plast: M0, M1, M2 in M3 (Engels in Sauer 2010, 419–20).

Slika 6.2 temeljna 4-plastna hierarhija meta-modela po standardu MOF  
  
vir: (Engels in Sauer 2010, 420)

**M0** predstavlja plast izvajanja in predstavlja področje uporabe (razvoj programske opreme). V domeni IM plast M0 predstavlja temeljne objekte, ki se proizvajajo tekom življenjskega cikla razvoja programske opreme (Engels in Sauer 2010, 419–20).

**M1** je plast modeliranja, ki predstavlja model MIPO (Engels in Sauer 2010, 420). V tej plasti so definirani konstrukti in procesni modeli (Jablonski 2010, 400). M1 se izražajo v realnosti kot lastne instance v M0 (Engels in Sauer 2010, 420).

**M2** je plast kjer se izvaja meta-modeliranje (Engels in Sauer 2010, 420). Konstrukti so glede na domeno uporabe definirani tukaj. Definirajo jih procesni jeziki na nivoju domenskega[[10]](#footnote-10) in abstraktnega[[11]](#footnote-11) modela. Abstraktni služi za podajanje osnovnih funkcionalnosti jezikov modeliranja kot sta potek in kontrola podatkov, domenski model pa predstavlja dejanski jezik modeliranja. Elemente nivoja M2 prestavljajo primitivne procesne entitete. Definiramo jih s pomočjo generičnih elementov modeliranja (škatle, loki), ki jih podaja plast M3 (Jablonski 2010, 400).

**M3** predstavlja najvišjo plast. Tukaj so definirani meta-meta-modeli (modeli meta-modelov). Namenjeni so za opisovanje meta-modelov v plasti M2 (Engels in Sauer 2010, 420).

Inženiring se dogaja v treh domenah: IMM[[12]](#footnote-12), IM in razvoj programske opreme. Vsaka od domen odgovarja različni stopnji abstrakcije, katere so izražene s hierarhijo plasti prikazani na sliki 6.3. Domene zahtevajo opravljanje različnih nalog za proizvajanje želenih produktov na posameznih nivojih meta-modela. Te naloge se izvajajo z dodelitvijo vlog kot kaže slika 5.5. Inženir meta-metod je odgovoren za definiranje meta-metode za IM na nivoju M2. To predstavlja domeno IMM. Definirano meta-metodo nato inženir metod aplicira v domeno IM z namenom izdelave stabilne MIPO za nivo M1. Izdelano MIPO lahko razvijalci uporabijo v domeni razvoja, ki ga predstavlja nivo M0 (Engels in Sauer 2010, 422).

Slika 6.3 dodeljene vloge za izdelavo delovnih produktov na različnih nivojih hierarhije meta-modela

Vir: (Engels in Sauer 2010, 423)

6.2.1 METODA METAME

Velika pomanjkljivost dosedanjih meta-metod je odsotnost definicij opravil in procesov, ki specificirajo izgradnjo MIPO z uporabo meta-modeliranja. Čeprav večina pristopov podpira nekakšno integracijo produktnih in procesnih fragmentov se le ta dogaja na visokem nivoju abstrakcije. Zaradi tega niso sposobni tvoriti kompleksne vzorce medsebojno povezanih strukturnih in vedenjskih modelov (Engels in Sauer 2010, 413).

Na podlagi temeljev IM, navedenih pomanjkljivosti in standardov meta-modeliranja so strokovnjaki razvili metodo MetaME[[13]](#footnote-13), ki je meta-metoda za razvoj MIPO (Engels in Sauer 2010, 424). Metoda sledi ideji izgradnje novih MIPO pred modifikacijo starih velikih procesov (Steenweg, Kuhrmann, in Méndez Fernández 2014, 14). Za potrebe izgradnje definira jezik za opis metode (produktna perspektiva) in proces, ki se uporablja za razvoj programske opreme (procesna perspektiva) (Engels in Sauer 2010, 424–25). MetaME metoda je nastala z združitvijo produktnega in procesnega modela, ki združujeta njima sorodne perspektive (Engels in Sauer 2010, 425). Inženir mora v produktnem modelu definirati katere artefakte je potrebno ustvariti in kako so ti artefakti med seboj povezani. Medtem mora v procesnem modelu definirati postopek izdelave artefaktov, kaj je potrebno narediti in kdo je za to odgovoren. Produktni model tako razumemo kot model artefaktov, procesni model pa kot model poteka dela in opravil (Engels in Sauer 2010, 413–14). Nato se ta dva modela združita v enoten proces. Ta združitev tvori MIPO, katera služi razvijalcem v praksi.

Slika 6.4 temeljni proces meta-metode za izgradnjo metode inženiringa programske opreme



Vir: (Engels in Sauer 2010, 430)

Slika 6.4 prikazuje temeljni proces meta-metode za izdelavo MIPO, ki se deli na:

* Definiranje domene in njenih disciplin,
* izdelava domenskega modela iz konceptov,
* izbira notacije,
* definicija tipov artefaktov,
* izgradnja procesnega modela,
* izbira orodij, tehnik in pripomočkov.

V prvem koraku moramo definirati domeno uporabe metode in njene discipline (Engels in Sauer 2010, 429). Lahko predstavljajo različne nivoje abstrakcije, poglede ali parcialne modele disciplin. Nato je zgrajen model iz konceptov ali temeljnih aktivnosti domene, katere so organizirane na podlagi disciplin (Engels in Sauer 2010, 429-30). Ta model predstavlja produktni model meta-metode. Sledi izbira notacije, ki skrbi za primerno predstavitev konceptov. V tem koraku je potrebno identificirati jezike, podjezike in elemente jezika. Navadno izbiramo med jeziki modeliranja (Engels in Sauer 2010, 430). Naslednji korak predstavlja definiranje tipa artefaktov. V tem koraku se jeziki in njegovi elementi pripišejo konceptom z namenom izražanja njihovih lastnosti. Medtem, ko domenski model predstavlja semantično domeno (pomen) konceptov, jeziki predstavljajo sintaktično domeno (oblika). Hierarhiji domenskega modela konceptov in artefaktov morata biti kompatibilna. Temu sledi definiranje procesnega modela pri katerem je potrebno definirati aktivnosti, ki so potrebne za izpolnitev nalog. Procesni model je strukturiran iz aktivnosti, mejnikov in elementov nadzora toka. Zadnji korak predstavlja izbira orodij, tehnik in pripomočkov skupaj s koncepti uporabe, ki so potrebni za usmerjanje in poenostavitev izgradnje artefaktov (Engels in Sauer 2010, 431).

6.3 SESTAVA METODE

Ta poglavje zaključujemo z izbiro in inženiringom metode, ki jo bomo uporabili pri izgradnji procesa oz. širše MIPO. Sedaj se podajamo v vlogo inženirja metode, ki ga predstavlja prva tretjina v sliki 6.3. Metodo bomo sestavili iz AM, MetaME metod, ki so bile predstavljene. Ko kaže slika 6.6 bo metoda MetaME predstavljala hrbtenico osnovnega procesnega toka razvoja video iger, AM pa bo s svojimi praksami služila za podporo iterativnim, inkrementalnim in ostalim konceptom agilnih praks.

Slika 6.6 izboljšan temeljni proces meta-metode z uporabo Agilnega modeliranja



Vir: (Ambler 2002, 10)

7 IZGRADNJA PROCESA

Izdelava procesa pomeni definiranje procesnega modela, ki predstavlja zgradbo sistema (Jablonski 2010, 394). Potrebno je definirati zbirko elementarnih primitivnih procesnih entitet iz katerih je mogoče zgraditi prilagojen procesni model, ki omogoča apliciranje v različne domene (Jablonski 2010, 396). Našo domeno predstavlja razvoj video iger, ki je hkrati prvi korak v procesu izgradnje MIPO po MetaME.

7.1 DEFINIRANJE DOMENE IN DISCIPLIN

Domeno predstavlja razvoj video iger katere discipline so povezane z vlogami in odgovornostmi. Discipline so lahko predstavljene tudi z različnimi nivoji abstrakcije, ki lahko predstavljajo korake ali faze procesa (zahteve, analiza, razvoj, itd.) (Engels in Sauer 2010, 429–30). Osnovne faze pri razvoju programske opreme predstavljajo analiza, načrtovanje, kodiranje in testiranje (Ramadan in Widyani 2013, 95). Faze, ki jih posamezni avtorji navajajo so prikazane v PRILOGA B. Na podlagi literature smo konsolidirali faze razvoja iger in jih kronološko razvrstili: formalizacija problema, razvoj koncepta, pred produkcija, produkcija, testiranje, post produkcija.

7.2 IZGRADNJA DOMENSKEGA MODELA

Resne igre potrebujejo posebno fazo v razvoju. Aslan in Balci (2015) definirata formulacijo problema kot prvo fazo pri razvoju resnih iger. Ta faza definira probleme različnih domen in načine reševanja, ki privedejo do efektivnega učenja na podlagi iger.

Za drugo fazo definirata generacijo idej (Aslan in Balci 2015, 311), ki pa predstavlja prvo fazo po Bates (2004) in Novak (2012). Widyani (2013) jo imenuje iniciacija. V tej fazi je predstavljena ideja igre v pisni obliki (Novak 2012, 352). Čeprav Unger in Novak (2011) označujeta pred produkcijo za prvo fazo pa navajata, da se začne po uspešno sprejetem konceptu. Zato bomo pred produkcijo kronološko postavili za fazo razvoja koncepta.

Pred produkcija je faza v kateri se sprejme odločitev kakšna bo igra in kako bo izdelana (Unger in Novak 2011, 176). Predstavlja fazo načrtovanja (Novak 2012, 353). Vrhunec pred produkcije predstavlja dokaz koncepta, ki ga predstavlja prototip. Namen je dokazati zmožnost premagovanja tehničnih zahtevnosti (Bates 2004, 88). Po potrditvi prototipa se lahko začne produkcija, ki predstavlja fazo razvoja igre (Novak 2012, 358).

Produkcija je kompleksna in časovno potratna. Za uspešno izvajanje faze je priporočena fragmentacija opravil v manjše naloge, katerim se strogo sledi. Priporočen je tedenski pregled napredka (Bates 2004, 212). Čeprav se testiranje v smislu produkcijskega testiranja dogaja tudi v fazi produkcije se v industriji iger ločijo posebne faze testiranja. Ko je razvoj igre tako daleč, da je mogoče igro igrati od začetka do zaključka se začnejo faze testiranja (Novak 2012, 359).

Testiranje se deli na Alfa in Beta faze. V razvoju mobilnih iger sta fazi navadno združeni (Unger in Novak 2011, 176). Alfa faza predstavlja točko na kateri je mogoče testirati celotno igranje. Testna ekipa v tej fazi testira vse module igre, zapiše defekte v bazo in ustvari testni načrt. Ob končani alfa fazi morajo biti narejeni naslednji elementi:

* celotno igranje,
* primarni jezikovni paket,
* osnovni uporabniški vmesnik z dokumentacijo,
* kompatibilnost z definiranimi strojnimi in programskimi konfiguracijami,
* test minimalnih zahtev,
* test kompatibilnosti vmesnikov (ali vsaj večino),
* nadomestni primeri zvoka in umetnin,
* test več igralske funkcionalnosti (če je implementirana),
* osnutek igralnega priročnika (Novak 2012, 359).

Beta faza je namenjena popravilu defektov in implementaciji vseh sredstev (zvok, umetnine) v igro. S tem se popolnoma konča produkcijski proces. Namen faze je stabilizacija projekta in eliminacija večine ali vseh defektov pred izdajo. Po končani beta fazi morajo biti končani naslednji elementi:

* koda,
* vsebina,
* jezikovni paket,
* igralna navigacija,
* uporabniški vmesnik,
* programska in strojna kompatibilnost,
* zvok in umetnine,
* vodnik igranja. (Novak 2012, 360).

Beta faza se deli tudi na zaprto in odprto beta fazo. Za zaprto beta fazo velja privatno testiranje pri kateri se razvoj fokusira na poliranje igre. Odprta faza pa je navadno uporabljena pri igrah ki vsebujejo spletne komponente. V tej fazi se preveri delovanje strežnikov, uravnoteži igranje in identificira defekte (Levy in Novak 2009, 54–55).

Ko igra prestane beta fazo sledi njena izdaja. V tej fazi ravnateljstvo opravi še en pregled produkta in pridobi seznam prisotnih defektov. Po pregledu sledi izdaja produkta na trg. V industriji iger je za to fazo pogost izraz ''zlata'' (Novak 2012, 362).

Po fazi izdaje sledi post produkcijska faza. Faza predstavlja nadaljevalno produkcijo, ki jo Fullerton (2008) imenuje tudi vzdrževanje. Slika 7.1 tako predstavlja domenski model, ki je zgrajen iz razvojnih faz in hkrati predstavlja podlago za procesni model.

Slika 7.1 domenski model na podlagi razvojnih faz



Vir: lasten

7.3 IZBIRA NOTACIJ

V tem koraku bomo definirali jezik uporabe, ki nam bo podal notacije. Jezik, ki ga bomo uporabili predstavlja UML z notacijami v tabeli 7.2. Z notacijami bomo povezali stanja in artefakte v procesni model.

Tabela 7.2 prikaz izbranih notacij UML jezika za definiranje procesa

|  |  |  |
| --- | --- | --- |
| NAZIV NOTACIJE | NAMEN | NOTACIJA |
| ZAČETNO STANJE | STANJE, KI NAKAZUJE ZAČETEK AKTIVNOSTI |  |
| IZBIRA | STANJE, KI OMOGOČA IZBIRO |  |
| TRANZICIJA | PRIKAZUJE SMER TRANZICIJE |  |
| STANJEs | PREDSTAVLJA STANJE |  |
| VILICE ALI ZDRUŽITEV | DELI DEJAVNOSTI NA VEČ TRANZICIJ ALI ZDRUŽUJE V ENO |  |

Vir: (Rumbaugh, Jacobson, in Booch 2004, 40–90)

7.4 DEFINIRANJE ARTEFAKTOV

Kot smo spoznali v petem poglavju artefakte sestavljajo produkti, dokumenti in diagrami. Artefakte smo razvrstili na podlagi literature in so zbrani v PRILOGI C.

Če se razvoj nameni izdelati učno igro mora skozi fazo, formalizacije problema. V tej fazi se kreira dokument specifikacije učnega problema (Aslan in Balci 2015, 310).

Nato sledi faza razvoja koncepta katere namen je definiranje osrednje funkcionalnosti igre, predstavitev grafičnega izgleda in zgodbe. Dokumente, ki jih razvoj koncepta proizvede so: višji koncept, predlog igre (''pitch doc'') in koncept (Bates 2004, 203). Vsak tip dokumenta, ki predstavlja koncept ima svoj namen, zato ni potrebno, da uporabimo vseh. Višji koncept služi za hitro predstavitev igre, ki bi denimo trajala do 30 sekund. Predlog igre je daljši dokument, ki služi za iskanje investitorjev. Tega Novak (2012) imenuje koncept. Pomaga ravnateljstvu oceniti izvedljivost ideje (Novak 2012, 382). Naslednji dokument Bates (2004) imenuje koncept in je obsežnejši, tudi do 20 strani in zajema opis žanra, igranja, funkcionalnosti, zgodbe, ciljne publike, izbiro platforme, pričakovane vire, analizo konkurenčnosti, analizo tveganja in povzetek (Bates 2004, 204). Novak (2012) ga imenuje predlog igre. Namen predloga igre je predstavitev igre podjetju ali partnerju. Ta dokument se bolje posveti zgodbi in opisu likov (Novak 2012, 387). Zaradi zagotavljanja konsistence bomo razvoj dokumenta koncepta razdelili v tri faze. V fazo izdelave inicialne ideje (višjega koncepta), kateremu sledi dokument koncepta igre (pitch) in izdelava predloga igre.

Predlog igre podpira prvi prototip[[14]](#footnote-14) igre. IGDA[[15]](#footnote-15) ga imenuje demo in predstavlja najpomembnejši material, ki ga lahko proizvedemo v tej fazi (Fullerton 2014, 486). Velja za dokaz delovanja koncepta in zmožnost premagovanja tehničnih zahtevnosti (Bartle 2003, 88). Če je dokaz koncepta uspešen se začne faza načrtovanja ali pred produkcija (Novak 2012, 353). Cilj je izdelava načrta igre, umetniške biblije, definiranje produkcijske poti in stvaritev projektnega načrta. Načrt igre je najobsežnejši dokument v razvoju igre. Novak (2012) navaja, da se ta dokument konstantno posodablja. Schell (2008) pa dodaja, da se občasno posodablja in je po navadi neurejen. Na polovici projekta se ga navadno opusti, saj vsebuje igra sama vse pomembne podrobnosti (Schell 2008, 382). Na podlagi načrta igre tehnična ekipa izdela tehnični dokument. Ta dokument predstavlja produkcijsko pot, ki vzpostavi načrt kako se bo razvoj premaknil od koncepta do programske opreme (Novak 2012, 393). Nato sledi izdelava načrta projekta, ki se začne izdelovati po definiranju grobih nalog, ki jih podaja tehnični dokument. Ta dokument vsebuje načrt porabe virov, proračun, časovni načrt, in mejnike za pomoč pri spremljanju napredovanja projekta (Novak 2012, 394).

Izdelava umetnikove biblije se začne v pred produkciji. Služi za definiranje umetniškega sloga igre (Bates 2004, 208). Ker je odvisna od žanra se začne po načrtu igre (Novak 2012, 393). Umetnikova biblija se navadno gradi iterativno tudi v času produkcije, ker se je v fazi pred produkcije še ne potrebuje (Bartle 2003, 87-88).

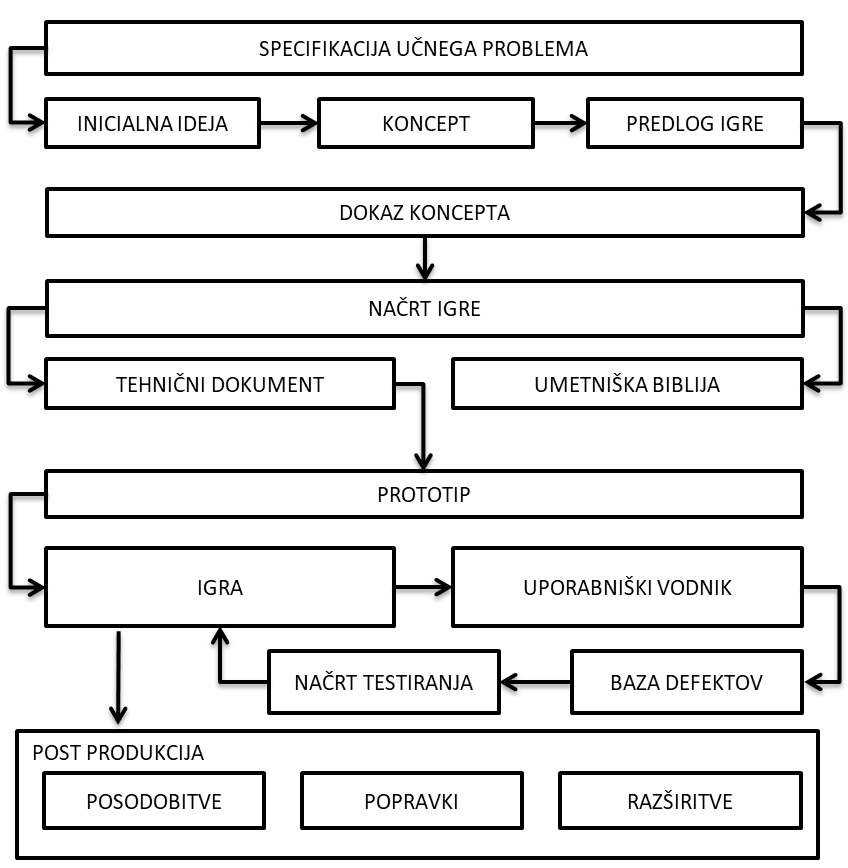
Zadnje dejanje pred produkcijske faze predstavlja izdelava prototipa (Bates 2004, 207–8). Ta artefakt je lahko analogne ali digitalne oblike. Pred izdelavo digitalnega je dobra praksa izdelava enostavnega analognega prototipa katerega namen je zagotoviti, da je igralna mehanika zabavna in prepričljiva. Končan prototip je lahko dokaz delovanja produkcijske poti od ideje do realizacije (Novak 2012, 354). Kot navajata Levy in Novak (2009) se lahko produkcija začne z izdelavo prototipa, ki je groba prestava igre in vsebuje primer igranja.

Nato sledi faza produkcije. V času produkcije se proizvajata dva artefakta. Sama igra ali njen fragment (prototip) in uporabniški vodnik. V praksi je priporočljivo sprotno dopolnjevanje priročnika, vzporedno z razvojem. Vodnik vsebuje razlago igre, vodnika po namestitvi, hiter začetek in detajlno razlago funkcionalnosti uporabniškega vmesnika (Rucker 2002, 6).

Produkciji sledi faza testiranja. Ta faza predstavlja zagotavljanje kakovosti[[16]](#footnote-16) in skrbi, da igra zadovoljuje vse potrebe preden se izda na trg (Levy in Novak 2009, 57). V tem koraku se izdelata podatkovna baza defektov in načrt testiranja. Po testiranju sledi faza post produkcije. V tej fazi se kot artefakti pojavijo popravki, posodobitve in razširitve. V tej fazi so lahko izdane različne verzije programa, ki služijo kot izboljšave. Te verzije so brezplačne in so ustvarjene z apliciranjem popravkov na originalno različico izdaje. Popravki, so lahko aplicirani tudi z namenom reševanja ostalih produkcijskih defektov. Posodobitve izboljšujejo originalno izdajo igre. Te so večinoma ustvarjene z namenom podaljšanja življenjske dobe igre. Zadnji artefakt v razvoju predstavljajo razširitve. Včasih delujejo kot samostojne igre ali pa potrebujejo originalno igro za delovanje (Novak 2012, 365).

Slika 7.3 spodaj prikazuje vse dokumente, diagrame ali produkte, ki predstavljajo model artefaktov, ki je ustrezno kronološko razvrščen. Podjetja imajo lahko različne standarde dokumentacije. Nekatera ustvarijo več dokumentov, druga manj. Navadno je dokumentacija obratno proporcionalna stopnji zaupanja založnika in izkušenosti razvojne ekipe. Ne glede na izkušnje ekipe, dobra dokumentacija pripomore k boljšem razvoju (III 2004, 319).

Slika 7.3 artefakti pri razvoju video iger



Vir: lasten

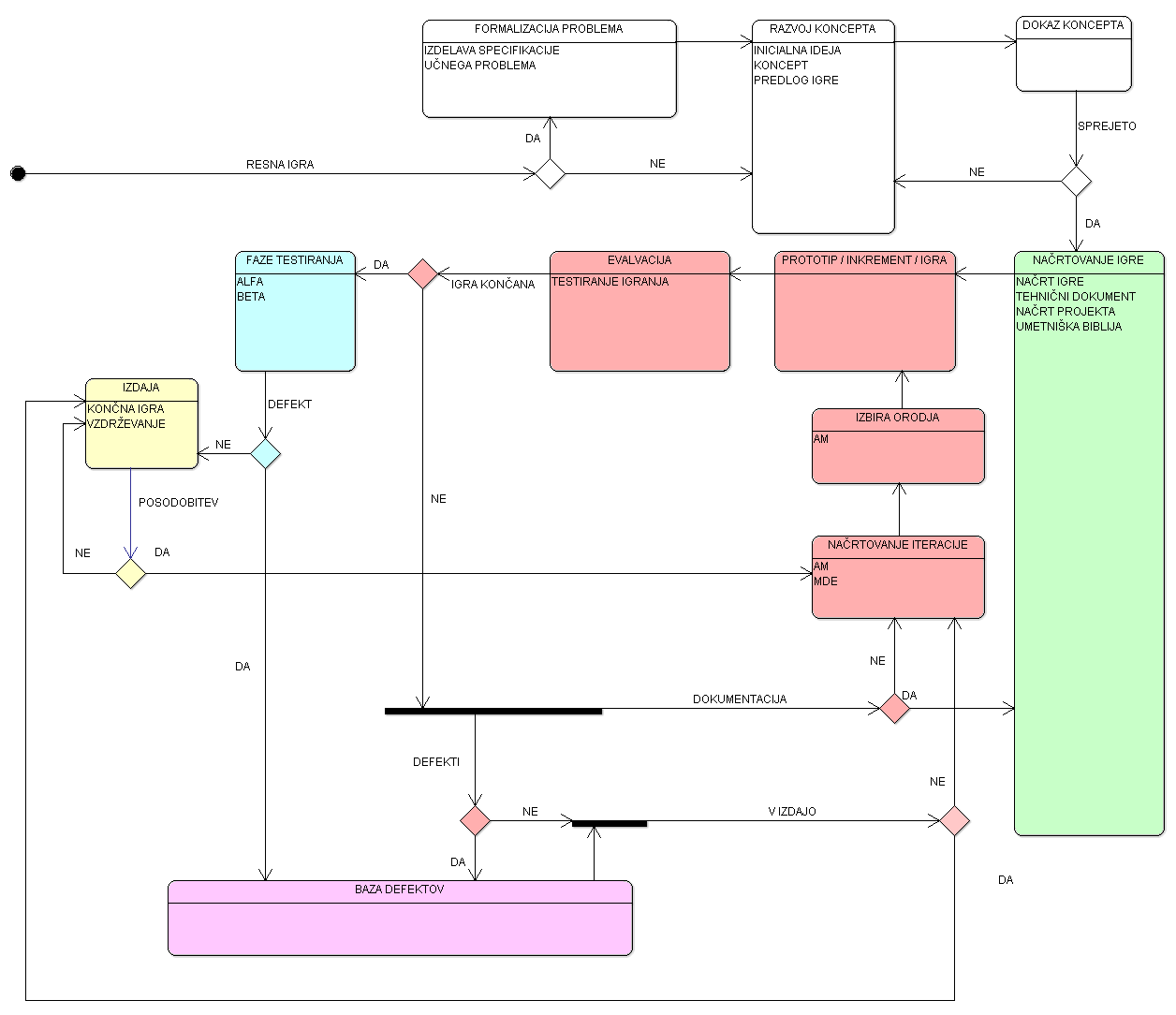
7.5 DEFINIRANJE PROCESNEGA MODELA

Slika 7.4 definira procesni model za video igre. Bela stanja predstavljajo fazo razvoja koncepta, ki se konča z dokazovanjem koncepta. Slednjega predstavlja prototip in velja za poenostavljeno različico igre. Oblikovalci iger jih uporabljajo za testiranje funkcionalnosti (Adams 2013, 49). Uporaben je tudi za dokazovanje izvedljivosti idej, saj predstavlja delujoči model ideje. Z njim si omogočimo formalizacijo ideje in izolacijo problemov (Fullerton 2014, 197). Prvi prototip, ki služi za dokazovanje izvedljivosti ideje je prvi mejnik s katerim se konča prva faza razvoja. V tem procesnem modelu se prototip pojavi že zgodaj v konceptualni fazi, ki dokazuje delovanje ideje preden se začne dolga faza načrtovanja. Ta lahko traja več mesecev (Adams 2013, 49).

Proces se nadaljuje v zeleno stanje, ki predstavlja pred produkcijo. Je proces, ki si ga je industrija iger sposodila od filmske industrije. Ta se navadno konča, ko je investitor že videl igralno različico igre in je zadovoljen z ekipo in delom. Če projektu poda zeleno luč se začne produkcijska faza (Adams 2013, 46).

Produkcijska faza se odvija med fazami načrtovanja, prototipa in evaluacije (Novak 2012, 367). Fullerton (2014) jo opisuje kot iterativni proces med testiranjem igranja, evalvacije in revidiranja. V našem modelu se testiranje igranja izvaja v stanju evalvacije. Po načrtovanju se izvede izgradnja prototipa. V fazi evalvacije ekipa igro testira in se odloči ali se vrne v fazo načrtovanja in nadgradi prototip. Ta proces se izvaja dokler igra ne predstavlja več prototipa ampak končno igro. Vsaka iteracija mora imeti svoj cikel razvoja skupaj z načrtovanjem potreb, artefaktov in urnika (Novak 2012, 367), zato ima model na sliki 7.4 v produkcijski fazi dva stanja (načrtovanje iteracij, izbire orodij), ki vsebujeta prakse in orodja, ki podpirajo te potrebe. Stanja produkcije so označena z rdečo barvo in se odvijajo iterativno dokler igra ni končna. Vzporedno se v produkciji odvijata tudi procesa posodabljanja baze defektov in dokumentacije. Baza defektov je označena z vijolično barvo in predstavlja aktivnost, ki se izvaja v več fazah (produkcija, testiranje, post produkcija).

Fazo testiranja predstavljajo modra stanja. V fazo testiranja preidemo, ko igra predstavlja končni izdelek ali če nas na to obvezujejo pogodbene obveznosti s strani investitorja. Ta faza se deli na alfa in beta fazo testiranja. Alfa stremi k zaklepanju funkcionalnosti medtem, ko se beta faza osredotoča na poliranje igre in reševanje defektov (Levy in Novak 2009, 52–53). V obeh faza se proces nadaljuje iterativno dokler obstajajo defekti ali dokler ti niso označeni za nepomembne (WNF[[17]](#footnote-17), NAB[[18]](#footnote-18)) (Levy in Novak 2009, 103), zato lahko v procesu pridemo do izdaje tudi preko baze defektov. Izdaja predstavlja zadnjo fazo v procesu in je v modelu označena z rumeno barvo. Izdelava popravkov po končani igri je skoraj neizogibno. Vzrok ne leži nujno v rani izdaji temveč v tisoče različnih strojnih konfiguracijah katere je nemogoče v celoti predvideti in testirati. Prav tako kot popravki predstavljajo posodobitve majhne projekte, ki zahtevajo načrtovanje mejnikov, testiranja in druge elemente dobrih praks (Bates 2004, 216). Procesni model to zagotavlja, saj se ob potrebi po posodobitvah (nadgradnje, popravki, razširitve) ponovno začne iterativni proces razvoja. Proces razvoja igre se tako nikoli ne konča. Razvijalci kljub izdanem produktu na podlagi povratnih informacij objavljajo popravke in včasih celo dodajo funkcionalnosti (Fullerton 2014, 421).

7.4 Procesni model za video igre  
  
Vir: lasten

7.5 IZBIRA ORODIJ

Izbira se odvija v času produkcije ali post produkcije (posodobitve) pred vsakim začetkom iteracije razvoja. Ta korak omogočata agilno modeliranje in modelno usmerjen inženiring v stanjih načrtovanja iteracije in izbire orodij. Orodja niso determinirana vendar se izbirajo po potrebi projekta ali same naloge. Tukaj se tudi izrazi v praksi, da se z aplikacijo AM v proces modelira več kot prej. V vsakem koraku aplikacije AM je cilj pridobiti globlje razumevanje enega ali več perspektiv sistema in s tem prakso, ki nam ustreza (Ambler 2002, 9). AM nam ponuja različne prakse, ki so se razvile na podlagi osnovnih kategorij (pog. 6.1) znotraj posameznih agilnih procesnih modelov. Nekaj vidnejših praks posameznih procesnih modelov je predstavljenih v PRILOGA F. Te prakse nato uporabimo za podporo našemu temeljnemu procesu v kombinaciji s popolnim agilnim procesom (sl. 6.6) (Ambler 2002, 10).

V razvoju video iger obstajajo tudi neke osrednje usmeritve, ki omogočajo uspeh projektov. Potrebno je zagotoviti: načrt projekta, razumljiv tehnični in časovni načrt, razumevanje tveganj in znanje ponastavitve projekta v težavah (Bates 2004, 219), ki predstavljajo bolj tehnične usmeritve. Ne tehnične predstavljajo: vzdrževanje komunikacije (z ekipo in vodstvom), sledenje stroškom, vzdrževanje ekipnega duha in identitete (Bates 2004, 213). Vse te usmeritve lahko zadovoljimo z izbiro pravega orodja ali več njih. Če pa se orodje ne izkaže za pravo v naslednji iteraciji izberemo drugega.

Pri načrtovanju razvoja je priporočljiva fragmentacija večjih nalog v manjše obvladljive naloge (Bates 2004, 212). Za takšno opravilo je priporočljiva uporaba Scrum prakse definiranja zgodb. Kot smo spoznali pa je Scrum le orodje za upravljanje in ne vsebuje tehničnih aspektov vodenja projekta. Zato so najverjetneje podjetja Scrum metodologijo posvojila v različnih oblikah. Dr. Lennart E. Nacke za Novak (2012) navaja:

*V naši ekipi smo dodali samolepilne liste na tablo za spremljanje nalog, kar omogoča izogibanje presežku poročanja in vizualno predstavo napredka posameznikov ali njihovih težav vsem v razvoju. Dr. Lennart E. Nacke.*

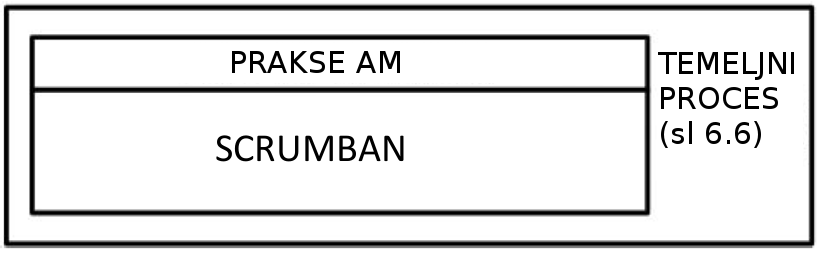
To nakazuje na uporabo hibrida Scrum in Kanban metodologije, ki jo predstavita Yilmaz in O'Connor. Imenovana Scrumban, omogoča, da so deležniki razvojnega procesa socialno povezani kar omogoča motivacijo v efektivno uporabo njihovih spretnosti (Yilmaz in O’Connor 2016, 237). Kanban dimenzija metodologije omogoča, da imamo v vsakem trenutku znanje koliko dela je v delovnem toku in na podlagi obremenitve lahko izračunamo stroške in napredek projekta, medtem, ko Scrum dimenzija omogoča temeljito vodenje projekta in fragmentacijo celotnega projekta na obvladujoče naloge z uporabo zgodb.

Načrtovanje lahko podpremo tudi z uporabo digitalnih programskih paketov. Ravnateljstvo že dolgo uporablja takšna orodja za administracijo individualnih ali skupine projektov. Orodja kot so: Scrumwise, Kanbanery, Yodiz, ZenHub, Leankit, Jira, Trello, so idealna za spremljanje nalog, saj omogočajo kreiranje Kanban tabel. Poleg tega omogočajo tudi komunikacijo med ekipami in integracijo z rešitvami za spremljanje izvorne kode kot je GitHub[[19]](#footnote-19). Poleg teh poznamo še Fat Panda, Pivotal Tracker, Active Collab, HacknPlan, Asana in Slack. Orodja lahko omogočajo tudi definiranje t.i. Gantt[[20]](#footnote-20) ali PERT[[21]](#footnote-21) grafov, ki zadostijo definiranjem časovnih načrtov.

Novi paketi takšnih orodij prinašajo prednosti, saj omogočajo upravljanje s tveganji, upravljanje z najboljšimi praksami, e-poštne notifikacije in kolaboracije. Potencialni kupci takšnih rešitev potrebujejo način s katerim lahko izberejo ustrezen programski paket. Za lažjo odločitev lahko uporabimo analitični hierarhični proces[[22]](#footnote-22), ki omogoča izbiro paketa na podlagi teorije izbire po Saatyu. Primarna naloga postopka je kvantifikacija relativnih prioritet za podano zbirko alternativ (PRILOGA E) na podlagi sodb, ki jih poda odločevalec (Ahmad in Laplante 2006, 76). Kriteriji se lahko spreminjajo in predlagani predstavljajo le ilustracijo procesa izbora. Kriterije je potrebno definirati na podlagi hierarhije, ki predstavlja za nas najbolj primerno orodje (Ahmad in Laplante 2006, 81).

V času razvoja je nevarnost, da projekt zaradi količine defektov postane neobvladljiv. Zavoljo tega obstajajo rešitve katere omogočajo spremljanje različic izvorne kode in takojšnjo ponastavitev na delujočo različico. Programska oprema kot so igre izstopajo predvsem zaradi količine delov iz katerih so sestavljene. Celo enostavna igra lahko vsebuje tisoče izvornih datotek kode, zvoka ali slik. Zato se priporoča izbira rešitve, ki omogoča nadzor izvorne kode (McShaffry in Graham 2012, 111). Komercialne rešitve predstavljajo SourceSafe, Perforce in AlienBrain, odprtokodne pa Subversion, TortoiseSVN in Git (McShaffry in Graham 2012, 15–114).

Slika 7.5 najvišja abstrakcija procesnega modela



Vir: lasten

V našem primeru predlagamo uporabo Scrumban procesa kot pomožnega orodja osnovnemu procesu v kombinaciji z agilnimi praksami. Scrumban izboljša Kanbanov WiP z vpeljavo sistema vlečenja, ki omeji količino dela v teku (Yilmaz in O’Connor 2016, 242). Obremenitev razvoja je tako minimalna in dnevnik zaostankov fiksen. Ocena hitrosti dela tako ni odvisna od števila zgodb, ki jih razvoj proizvede ampak od dnevnika zaostankov. Prav tako model posodobi dnevne sestanke, katere nadgradi v pol strukturirane intervjuje. Posledično ti sestanki omogočajo ustvarjanje sistematičnih in primerljivih informacij (Yilmaz in O’Connor 2016, 242). V realnosti veliko razvojnih ekip zgreši pri napovedi trajanja projekta. Zavoljo tega se proti koncu produkcijskega cikla velikokrat pojavi čas krize[[23]](#footnote-23) (Novak 2012, 358). Z uporabo tega procesnega modela pridobimo na reševanju težav slabih časovnih napovedi razvoja. Poleg tega predlagamo uporabo XP prakse konstantno prisotnega vlagatelja in fiksnega 40 urnega tedenskega delavnika, ki še podpre reševanje kriznega časa in onemogoča preobremenitev zaposlenih.

1. Institute of Electrical and Electronic Engineers – IEEE. [↑](#footnote-ref-1)
2. (angl.) User Experiece-UX. [↑](#footnote-ref-2)
3. (angl.) Playtesting. [↑](#footnote-ref-3)
4. 35% manjkajočih vrednosti (Nesodelovanje). [↑](#footnote-ref-4)
5. Agile Manifesto. Manifesto for Agile Software Development. [↑](#footnote-ref-5)
6. SoftwareEngineering – Metamodel for Development Methodologies. [↑](#footnote-ref-6)
7. Software & Systems Process Engineering Meta-Model – SPEM. [↑](#footnote-ref-7)
8. Object Management Group – OMG. [↑](#footnote-ref-8)
9. Meta Object Facility – MOF. [↑](#footnote-ref-9)
10. (angl.) Domain Specific Process Meta Model – DSPMM. [↑](#footnote-ref-10)
11. (angl.) Abstract Process Meta Model – AMM. [↑](#footnote-ref-11)
12. Inženiring meta-metod. [↑](#footnote-ref-12)
13. Meta-Method for Software Engineering Methods. [↑](#footnote-ref-13)
14. (angl.) Proof of concept. [↑](#footnote-ref-14)
15. International Game Development Association. [↑](#footnote-ref-15)
16. QA- Quality assurance. [↑](#footnote-ref-16)
17. Will Not Fix, pomeni, da defekt ni dovolj pomemben. [↑](#footnote-ref-17)
18. Not A Bug, pomeni da ni defekt. [↑](#footnote-ref-18)
19. Git distribuiran sistem za nadzor različice kode. [↑](#footnote-ref-19)
20. Je grafikon, ki predstavlja časovni razpored projekta. [↑](#footnote-ref-20)
21. Program evaluation and review technique, je statistično orodje za analizo nalog, ki so vključene v razvoj. [↑](#footnote-ref-21)
22. AHP – Analytical Hierarchy Process. [↑](#footnote-ref-22)
23. (angl.) Crunch time. [↑](#footnote-ref-23)