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, popolnitev potreb, katere vodijo v zagon novega 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 in/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 (slika 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). Produktni fragmenti (artefakti) so ciljno orientirani in jih predstavljajo produkti, dokumenti, modeli, diagrami in dr.. Procesna dimenzija pa mora definirati postopke, ki so potrebni za izdelavo artefaktov in njihove izvajalce (Engels in Sauer 2010, 413). Procesni fragmenti (postopki) 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

Agilno modeliranje definira štiri prakse, ki podpirajo iterativni in inkrementalni pristop k modeliranju:

1. Apliciranje pravih artefaktov,
2. izdelava več vzporednih modelov,
3. iteracija do naslednjega artefakta,
4. izvajanje manjših inkrementov (Ambler 2002, 45).

Artefakte predstavljajo UML grafikon stanja, izvorna koda, diagram poteka podatkov, primeri uporabe in drugi. 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).

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 kot kaže slika 6.5 v plasti M1. Ta združitev tvori MIPO, katera služi razvijalcem v praksi.

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

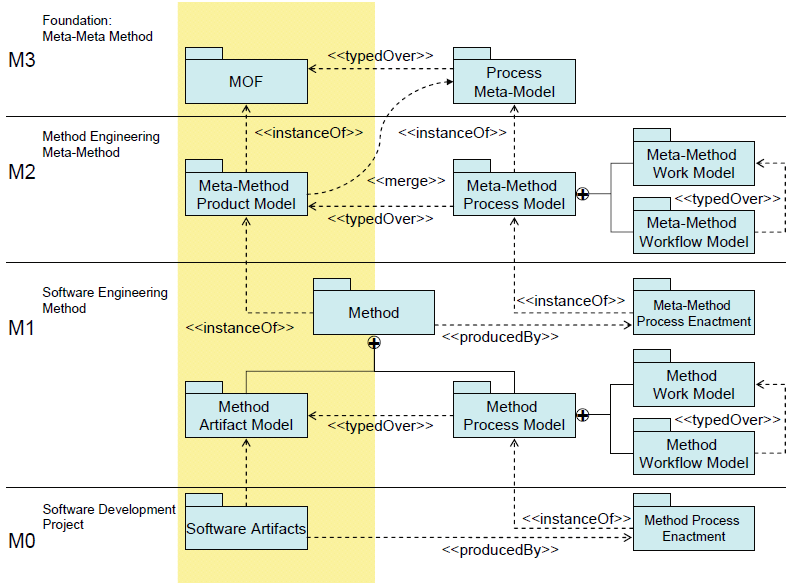


Vir: (Engels in Sauer 2010, 430)

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.

Slika 6.5 apliciranje pristopa meta-modeliranja za inženiring metod



Vir: (Engels in Sauer 2010, 426)

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. Proces je generalno 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).

6.2.2 MODELNO USMERJEN INŽENIRING

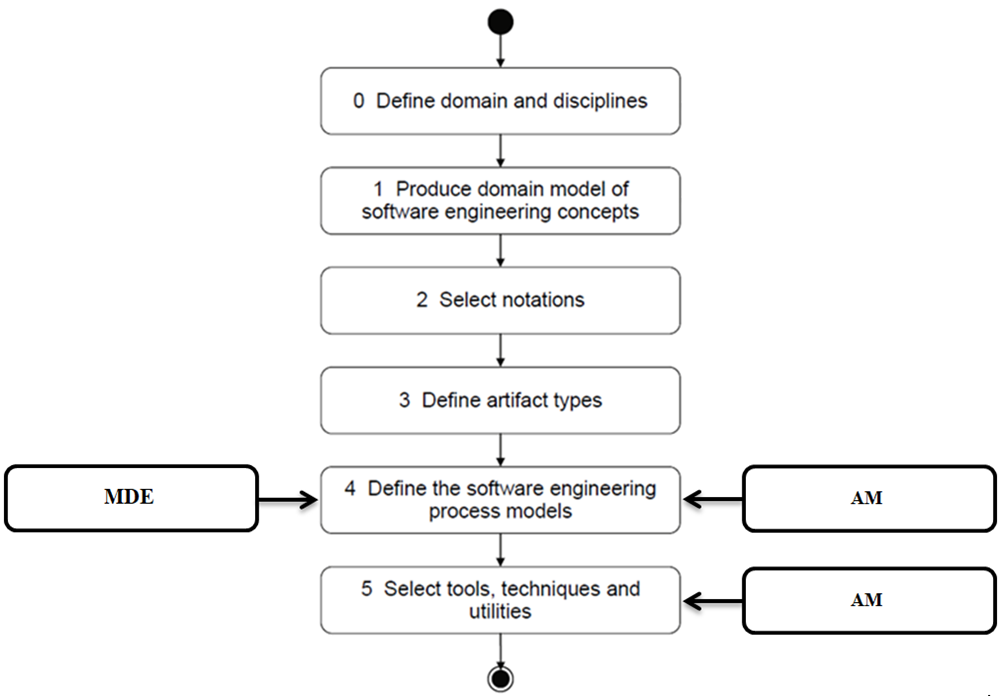
Cooper in Shaun predstavita modelno[[14]](#footnote-14) usmerjen inženiring (MDE) za izboljšanje razvoja resnih iger, ki sloni na temeljih meta-modela. MDE združuje elemente tradicionalnega oblikovanja iger, pedagoških vsebin in programskega inženiringa (Cooper in Scacchi 2015, 17). Avtorji pristopa MDE navajajo, da se ga lahko umesti v agilni, iterativni razvojni proces, z namenom neformalnega, pol formalnega ali formalnega opisa dela igre. Meta-model olajša razvoj visokokvalitetnih resnih iger, ker eksplicitno povezuje znanje in spretnosti z razvojem iger (Cooper in Scacchi 2015, 66). Razvojni proces z uporabo MDE je razdeljen na tri korake:

* Prvi korak je izgradnja neformalnega modela igre, ki predstavlja zgodbo, opise, ciljev, igranja in konceptov uporabniškega vmesnika,
* drug korak je transformacija neformalnega modela v pol formalnega z uporabo UML. V tem koraku se nadgradijo osnovni opisi igranja, zgodbe, grafike, animacije, glasbe, zvočnih efektov in ostale vsebine,
* tretji korak je transformacija pol formalnega modela v formalni model z uporabo XML[[15]](#footnote-15) jezika. V tem koraku je potrebno definirati shemo XML glede na potrebe in atribute, ki jih želimo uporabiti. XML nato predstavlja specifikacijo igre, ki jo lahko naložimo in testiramo (Cooper in Scacchi 2015, 16–17).

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 in MDE metod, ki so bile predstavljene. Temelje bo predstavljala metoda MetaME in bo predstavljala hrbtenico osnovnega procesnega toka razvoja video iger, AM in MDE pa bosta služili kot orodja za podporo iterativnim, inkrementalnim in ostalim konceptom agilnih praks.

Slika 6.6 izboljšan temeljni proces meta-metode za izgradnjo metode inženiringa programske opreme



Vir: (Engels in Sauer 2010, 426)

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. Novak (2012) posamezne vloge postavi v sklope: Produkcija, načrtovanje, kreativnost, kodiranje, zvok, testiranje in nadzor kvalitete. V delu Game Development Essentials: Mobile Game Development pa Unger in Novak vloge razdelita na: upravljanje, načrtovanje, umetnost, kodiranje, zvok, testiranje in nadzor kvalitete (Unger in Novak 2011, 182–86). Čeprav je delo Game Design Batesa starejše je opaziti, da se v razvoju iger niso zgodile večje spremembe v razdelitvi disciplin. Bates jih deli na produkcijo, načrtovanje, kodiranje, umetniško delo in testiranje (Bates 2004, 159–76). Od drugih razdelitev se razlikuje le zvok, ki ga Bates umešča med zunanje vire (Bates 2004, 183).

Discipline so lahko predstavljene z tudi 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).

Unger in Novak (2011) delita faze na pred produkcijo, produkcijo, Alfa & Beta[[16]](#footnote-16), Gold[[17]](#footnote-17) in post produkcijo. Novak (2012) podrobneje na: koncept, pred produkcijo, prototip, produkcijo, Alfa, Beta, Gold in post produkcijo. Blitz studio za razvoj iger definira šest korakov: nagovor[[18]](#footnote-18), pred produkcija, produkcija, alfa, beta in master. Doppler Interactive definira: načrtovanje, iteracija med razvojem in evalvacijo, testiranje, pregled (ki se lahko vrne v fazo načrtovanja in izdaja. Ramadan in Widyani definirata načrtovanje, prototip, produkcijo in testiranje za osnovne faze. Na podlagi njune analize literature pa konsolidirata faze različnih avtorjev in predlagata naslednje faze: iniciacija, pred produkcija, produkcija, testiranje, beta, izdaja (Ramadan in Widyani 2013, 100). Zelo razčlenjene korake nam razdelita Babu in Maruthi. Faze, ki jih definirata si sledijo: razvoj zgodbe, razvoj skripte, študija izvedljivosti, promocijski demo, načrtovanje, načrtovanje postavitve, modeliranje, teksturiranje, animiranje, podrobno načrtovanje, kodiranje, testiranje, razhroščevanje, integracija, testiranje igranja („Home - IJIRCCE::HIGH IMPACT FACTOR JOURNAL: 6.577“ 2017).

Ker želimo ustvariti model tudi za potrebe resnih iger moramo identificirati določene korake. Aslan in Balci predlagata naslednje faze za potrebe razvoja resnih iger: formulacija problema, izdelava ideje, načrtovanje igre, razvoj zahtev, arhitektura, načrtovanje programske opreme, programiranje, integracija, izdaja, učenje, povratna informacija (Aslan in Balci 2015, 309).

7.2 IZGRADNJA DOMENSKEGA MODELA

Slika 7.1 prikazuje prikaz disciplin, ki jih navajajo različni avtorji. Discipline so obarvane glede na namen. Zvok spada med kreativne vsebine, zato se obarvan enako kot kreativnost in umetnost. Prav tako je testiranje del nadzora kvalitete, zato je enako obarvan. Na podlagi definiranja barv disciplin smo obarvali tudi tabelo (gl. PRILOGA B), ki prikazujejo posamezne faze razvoja. Faze spadajo v posamezne discipline vendar posedujejo časovno dimenzijo, ki določajo odvisnosti disciplin.

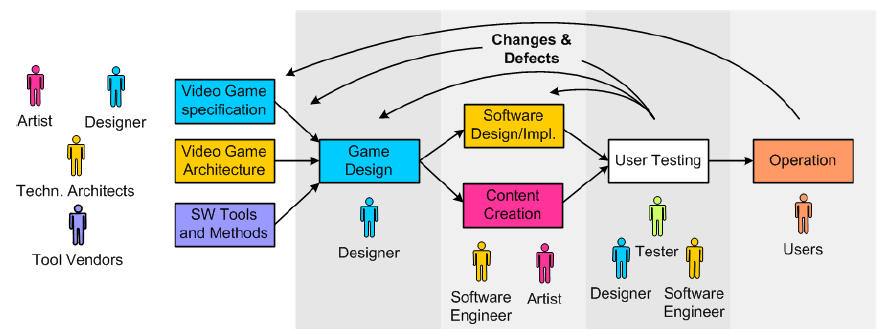
Slika 7.1 izpis disciplin po avtorjih

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| AVTOR | DISCIPLINE |  |  |  |  |  |  |
| Novak (2012) | PRODUKCIJA | NAČRTOVANJE | KREATIVNOST | KODIRANJE | ZVOK | TESTIRANJE | NADZOR KVALITETE |
| Unger in Novak (2011) | UPRAVLJANJE | NAČRTOVANJE | UMETNOST | KODIRANJE | ZVOK | TESTIRANJE | NADZOR KVALITETE |
| Bates (2004) | PRODUKCIJA | NAČRTOVANJE | UMETNIŠKO DELO | KODIRANJE |  | TESTIRANJE |  |

Vir: lasten

Ideja je, da na podlagi konsolidiranja vseh faz ustvarimo izpopolnjeni predlagani potek dela, kot ga prikazuje slika 7.2.

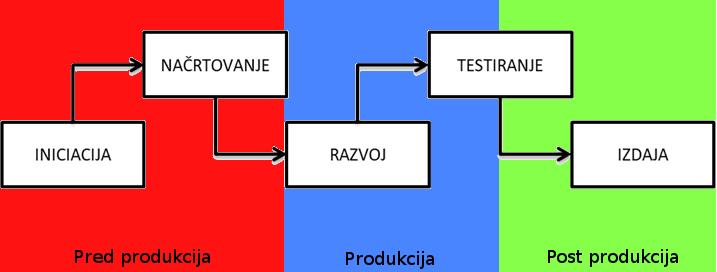
Slika 7.2 enostaven potek dela za razvoj iger



Vir: (Musil in dr. 2010, 5)

Slika 7.3 prikazuje konsolidirane faze, ki predstavljajo predlagani domenski model. S tem smo definirali procesno dimenzijo meta-modeliranja.

Slika 7.3 predlagani domenski model



Vir: lasten

7.3 IZBIRA NOTACIJ

V tem koraku bomo definirali jezik uporabi, ki nam bo podal notacije. Jezik, ki ga bomo uporabili predstavlja UML z notacijami v tabeli 7.4. Z notacijami bomo povezali stanja in artefakte v procesni model.

Tabela 7.4 prikaz izbranih notacij UML jezika za definiranje procesa

|  |  |  |
| --- | --- | --- |
| NAZIV NOTACIJE | NAMEN | NOTACIJA |
| AKTER | ZUNANJI UPORABNIK SISTEMA |  |
| ARTEFAKT | FIZIČNI KOS INFORMACIJE |  |
| ZAČETNO STANJE | PSEUDOSTANJE, KI NAKAZUJE ZAČETEK STANJA |  |
| KONČNO STANJE | STANJE, KI NAKAZUJE KONČANJE AKTIVNOSTI |  |
| TERMINACIJA | STANJE, KI PREKINJA IZVEDBO |  |
| IZBIRA | STANJE, KI OMOGOČA IZBIRO |  |
| TRANZICIJA | PRIKAZUJE SMER TRANZICIJE |  |
| STANJE | PREDSTAVLJA STANJE |  |

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

7.4 DEFINIRANJE ARTEFAKTOV

Ko smo spoznali v petem poglavju artefakte sestavljajo produkti, dokumenti in diagrami. Vsi omenjeni avtorji in artefakti so zbrani v PRILOGI C. Novak (2012) definira naslednje artefakte: koncept, predlog igre, načrt igre[[19]](#footnote-19), vodnik umetniškega sloga, tehnični dokument, projektni načrt in načrt testiranja. Bates (2004) artefakte označi z drugačnimi konstrukti: načrt igre, opis oblikovanja, specifikacija potreb, plan konfiguracije, uporabniški priročnik, načrt integracije testa in testiranja. Specifikacijo in plan konfiguracije lahko razumemo kot del tehničnega dokumenta. Izstopa le uporabniški priročnik, ki naj bi vseboval razlago delovanja igre, vodnika za namestitev in uporabo in celostno razlago uporabniškega vmesnika in njegovih funkcionalnosti. V praksi je priporočljivo sprotno dopolnjevanje priročnika, vzporedno z razvojem (Rucker 2002, 6). Rucker (2002) pa navaja specifikacijo, časovni načrt, načrt oblikovanja in dokumentacijo kot osnovne dokumente (Rucker 2002, 36). Schell (2008) razdeli dokumente glede na šest osnovnih skupin (sl. 7.5) znotraj katerih se le ti ustvarjajo. Skupine so: oblikovanje, pisanje, upravljanje, inženiring, umetnost in igranje. Znotraj teh skupin definira: Podrobneje dokumente razdeli tudi Rouse III (2004). Navaja, da se kot prvi dokumenti pojavijo konceptualni dokument, pitch ali predlog (III 2004, 308). Nato sledi konkurenčna analiza, ki se lahko kreira pred samim predlogom igre ali pa, ko je razvoj igre že potrjen. Sledi načrt igre, ki ga nekateri razvijalci označujejo za funkcionalno specifikacijo, pogosto pa je tudi odskočna deska za časovni načrt (III 2004, 309–10). Načrt igre navadno vsebuje tudi diagram poteka, lahko pa predstavlja svoj dokument. Diagram poteka predstavlja vizualno predstavo delovanja igranja in konsekvence igralčevih odločitev (III 2004, 311). Nato sledi zgodba in, če le ta vsebuje dialoge tudi dokument naracije (III 2004, 313). Naslednji pomemben dokument je umetniška biblija, ki lahko poleg usmeritev umetniškega sloga tudi tehnično dokumentacijo. Lahko narekuje število poligonov ali število slik na sekundo pri animacijah. Umetniška biblija lahko vsebuje tudi snemalno knjigo, ki služi za pomoč pri snemanju ali modeliranju prizorov. Sorodni načrtu igre je tehnični dokument. Medtem, ko načrt igre opisuje funkcionalnost igre, tehnični dokument definira načine njihove implementacije (III 2004, 317). Kot zadnje pa Rouse III (2004) omeni časovne, poslovne in marketinške dokumente, kateri se pogosto znajdejo v takšni ali drugačni obliki v načrtu igre. Zaradi svoje narave so bolj primerni za ljudi, ki so usmerjeni v prodajo in oglaševanje. Dokumente je potrebno posodabljati in je bolje, da so odgovornosti porazdeljene glede na usposobljenost (III 2004, 318). Adams definira višji koncept kot abstraktni dokument, ki služi za dokumentiranje idej. T.i. pitch predstavlja dokument analize igre, ki služi za predstavitev ideje naročniku. Načrt igre razdeli na oblikovanje osrednjega igralca, oblikovanje sveta in uporabniškega vmesnika. Ti dokumenti skupaj predstavljajo načrt igre, ki lahko vsebuje tudi diagram poteka delovanja in odločitev. Kar drugi avtorji niso opredelili je dokument napredovanja, ki definira napredovanje po stopnjah, če je igranje temu namenjeno. Prav tako definira tudi avdio dokument, ki je združen z dokumentom naracij. Kot zadnjega omenja scenarij igre, ki predstavlja definicijo temeljnega igranja igre (Adams 2013, 59–62).

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 dobra dokumentacija pripomore k boljšem razvoju (III 2004, 319). Slika 7.5 spodaj prikazuje konsolidacijo vseh dokumentov, ki predstavljajo model artefaktov. Prvi dokument predstavlja koncept, ki pomaga ravnateljstvu oceniti izvedljivost ideje (Novak 2012, 382). Predloga igre ali nagovor je naslednji dokument, ki predstavlja komponente igre bolj podrobno. Namen predloge igre je predstavitev igre podjetju ali partnerju. Ta dokument se bolje posveti zgodbi in opisu likov (Novak 2012, 387). Nadgradnja predloge igre je načrt igre. Temu sledijo tehnični dokument, dokument testiranja in dokument zvoka in naracij. Medtem ko so navedeni dokumenti navadno nastavljeni v pred produkcijskem času se umetnikova biblija navadno gradi iterativno v času produkcije (Bartle 2003, 87-88). Zadnji artefakt predstavlja prototip, ki velja za dokaz delovanja koncepta in zmožnost premagovanja tehničnih zahtevnosti (Bartle 2003, 88).

Slika 7.5 artefakti pri razvoju video iger



Vir: lasten

7.5 DEFINIRANJE PROCESNEGA MODELA

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. Model Driven Engineering – MDE. [↑](#footnote-ref-14)
15. Extensible markup language – XML. [↑](#footnote-ref-15)
16. Predstavljata stopnji testiranja pri čemer Beta sledi Alfa testiranju. [↑](#footnote-ref-16)
17. Kompletno zaključen projekt. [↑](#footnote-ref-17)
18. (angl.) Pitch. [↑](#footnote-ref-18)
19. (angl.) Game Design Document. [↑](#footnote-ref-19)