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 za metodo, ki predstavlja prvi korak. Sledi mu več-stopenjski razvojni proces. Po razvoju se metodo namesti v uporabo in preveri z namenom zagona novega evolucijskega cikla (Engels in Sauer 2010, 422).

IM se ukvarja s sistematično konstrukcijo metod inženiringa programske opreme (MIPO) (Engels in Sauer 2010, 412). 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). 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 (Engels in Sauer 2010, 411).

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 procesna in produktna perspektiva fragmentov metod. Produktni fragmenti so ciljno orientirani in jih predstavljajo produkti (fragmenti produkta), dokumenti, modeli, diagrami in dr.. Procesni fragmenti pa 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 delita metode na komponente. Komponenta je najmanjša smiselna entiteta metod in sestoji iz procesa, notacije in koncepta (Sunyaev, Hansen, in Krcmar 2009, 648).

Slik 5.1 življenjski cikel IM



Vir: (Engels in Sauer 2010, 422)

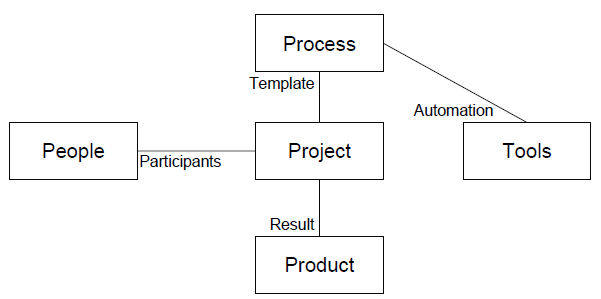
5.1 KOMPONENTE METOD

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). Po principu fragmentacije metod predstavljajo procesno dimenzijo metode, ki se deli na faze, aktivnosti in opravila (Sunyaev, Hansen, in Krcmar 2009, 647).

Notacije predstavljajo semantična, sintaktična in simbolna pravila za dokumentiranje. (Sunyaev, Hansen, in Krcmar 2009, 648).

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 (gl. PRILOGA A), nekateri pa se uporabljajo pogosteje: 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). Slika 5.2 prikazuje pet osnovnih konceptov, ki so jih podali avtorji unificiranega procesnega modela razvoja programske opreme (Engels in Sauer 2010, 416).

Slika 5.2 Osnovni koncepti unificiranega procesnega modela



Vir: (Engels in Sauer 2010, 416)

S procesi sta povezana dva termina, ki tvorita časovno perspektivo procesa razvoja programske opreme: cikel razvoja programske opreme in življenjski cikel razvoja programske opreme.

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).

Metode inženiringa programske opreme definirajo naslednje aspekte:

* 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 koncepte (gl. PRILOGA A); vloge ali ljudi, procese in aktivnosti (naloge), produkte ali artefakte, orodja (programska orodja ali uporabnosti) (Engels in Sauer 2010, 416).

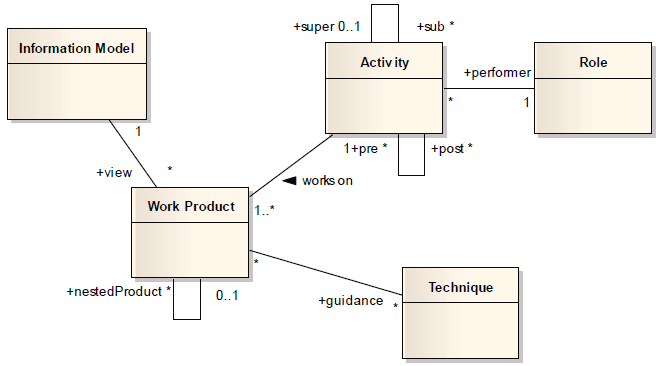
6 META-MODELIRANJE

Inženiring metod nam podaja različne pristope k inženiringu procesov (Engels in Sauer 2010, 412). Obetaven pristop naproti sistematičnemu in strukturiranemu razvoju programske opreme je uporaba tehnik meta-modeliranja za izgradnjo procesnih modelov (Engels in Sauer 2010, 419). V literaturi je definiranih več pristopov meta-modeliranja različnih avtorjev. Obstajata tudi dva standarda, ki uporabljata meta-modele za definiranje MIPO: ISO 24744:2007[[2]](#footnote-2) in SPEM[[3]](#footnote-3). 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).

Meta-model potrebuje pet osnovnih elementov za opis metode: aktivnosti, vloge, delovne produkte, tehnike in informacijski model (sl. 5.3) (Engels in Sauer 2010, 416).

Slika 5.3 temeljni koncepti inženiringa metod po Gurtzwiller



Vir: (Engels in Sauer 2010, 416)

Konzorcij OMG[[4]](#footnote-4) je v standardu MOF[[5]](#footnote-5) definiral štiri-plastno referenčno arhitekturo meta-modela na podlagi pogosto uporabljenih konceptov (Engels in Sauer 2010, 419). Kot je razvidno iz slike 5.4 se plasti delijo na plast: M0, M1, M2 in M3 (Engels in Sauer 2010, 419–20).

Slika 5.4 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, ki predstavlja model MIPO. Realnost je zmodelirana z jezikom modeliranja tako, da M0 predstavlja instance elementov plasti M1 (Engels in Sauer 2010, 420).

**M2** je plast kjer se izvaja meta-modeliranje. Vsebuje meta-modele kot so UML[[6]](#footnote-6) in SPEM. Ti definirajo jezike modeliranja kateri služijo za izgradnjo MIPO modelov (Engels in Sauer 2010, 420).

**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 metod se dogaja v treh domenah: IMM[[7]](#footnote-7), IM in razvoj programske opreme. Vsaka od domen odgovarja različni stopnji abstrakcije, katere so izražene s hierarhijo plasti prikazani na sliki 5.4. 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 5.5 dodeljene vloge za izdelavo delovnih produktov na različnih nivojih hierarhije meta-modela

Vir: (Engels in Sauer 2010, 423)

6.1 METODA METAME

Velika pomanjkljivost dosedanjih meta-metod je odsotnost definicij opravil in procesov, ki specificirajo izgradnjo MIPO na podlagi uporabe pristopa 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[[8]](#footnote-8), ki je meta-metoda za razvoj MIPO (Engels in Sauer 2010, 424). Metoda 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).

Procesni model je kompozit delovnih elementov (aktivnosti, naloge, faze) in toka dela, ki opredeljuje njihovo časovno sosledje (Engels in Sauer 2010, 425). f

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). Procesi sami so tudi programska oprema (Sutton 2011, 291). To pomeni, da lahko procese 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).

Programsko opremo lahko predstavimo s podatki, kar nam omogoča apliciranje entiteto-relacijskega[[9]](#footnote-9) jezika modeliranja. Entitete nam predstavijo podatke in relacije njihove medsebojne odvisnosti. Zaradi elementarne narave, ga je mogoče aplicirati na katerekoli podatkovne strukture. Veliko prednost uporabe pa se kaže v njegovi nezapletenosti (Jablonski 2010, 395).

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 celotnega sistema in prenos v delovno okolje (Satzinger, Jackson, in Burd 2011, 6).

Modeli so zgrajeni na podlagi inženiringa imperativov in inženiringa oblikovanja (Rolland in Ufr 1994, 1). Inženiring imperativov vključuje identifikacijo problemov ali potreb in oblikovanje specifikacije sistema. Na podlagi slednje inženiring oblikovanja poskrbi za izgradnjo in implementacijo v delovni sistem (Rolland in Ufr 1994, 2).

Procesni model ustreza načinu dela, ki ga predpisuje metodologija v uporabi. Model lahko razumemo tudi kot koncept načrta. Za oblikovanje takšnih načrtov je potrebna uporaba abstrakcije procesa, ki jo predstavlja meta model (slika 5.1). Pri uporabi procesnega modela v praksi je ta instanca meta modela (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)

Meta model predstavlja ključne aktivnosti, ki v razvojnem procesu razlagajo kdaj se kaj zgodi in zakaj (Rolland in Ufr 1994, 10).

Procesni model definira zbirko generičnih aktivnosti. Metodološkim inženirjem pomaga definirati način dela določene metodologije, ki jo je potrebno razviti. Način dela podane metodologije pa bo pridobljena z inicializacijo generičnih aktivnosti procesnega modela (Rolland in Ufr 1994, 10).

Inženiring imperativov sestavljajo situacije, odločitve, argumenti in dejanja. Meta model prepozna kontekst v katerem se pojavi odločitev, potreba po argumentaciji in njen izid. Nato poveže proces in končen izdelek s predstavitvijo imperativnih artefaktov kot vhodnih in izhodnih informacij procesa (Rolland in Ufr 1994, 11).

Agilno modeliranje definira štiri prakse, ki podpirajo iterativni in inkrementalni prostop 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 fokusirajo na efektivne ure razvoja kot pa njihovo število (Unger in Novak 2011, 180).

5.2 DEFINIRANJE KONCEPTOV

5.3 IZGRADNJA PROCESA

Eksekucija procesov je veliko več kot upoštevanje korakov po determiniranem vrstnem redu. Proces potrebuje podporo infrastrukture, ki omogoča njegove omejitve in hkrati dovolj fleksibilnosti, da ne ovira kreativnost uporabnikov (Jablonski 2010, 393). Izdelava procesa pomeni definiranje procesnega modela, ki predstavlja zgradbo sistema (Jablonski 2010, 394).

5.4 OPTIMIZACIJA PROCESA

Optimizacija pomeni razumevanje trenutnega procesa in njegovo spreminjanje z namenom povečanja kvalitete produktov, zmanjšanja stroškov in časa razvoja. Za optimizacijo se uporabljata zrelostni in agilni pristop (Sommerville 2010, 706).

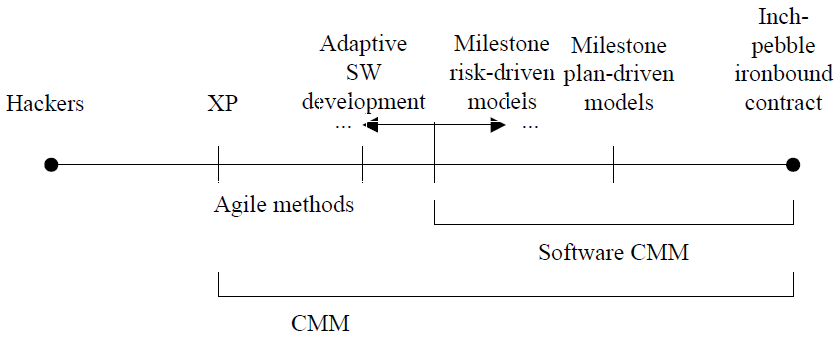
*Zrelostni pristop* je fokusiran na izboljšave procesov, vodenje projektov in implementacijo boljših inženirskih praks v organizacijo. Nivo zrelosti odraža obseg adaptacije tehničnih in vodstvenih praks v proces razvoja programske opreme. Cilj pristopa je izboljšanje kvalitete produktov in predvidljivost procesa (Sommerville 2010, 706).

*Agilni pristop* je fokusiran na iterativni razvoj s krčenjem presežkov v razvojnem procesu. Temeljne karakteristike agilnih metod so hitre izdaje funkcionalnosti in visoka odzivnost na spremembe (Sommerville 2010, 706).

Zrelostni pristop je zakoreninjen v načrtno usmerjenem razvoju in navadno proizvede veliko presežkov v smislu, da definira več aktivnosti kot je potrebno. Agilni pristop pa se fokusira na kodo in se namensko izogiba formalnostim in dokumentaciji (Sommerville 2010, 706). Za večje projekte in kompleksne sisteme je priporočena uporaba zrelostnega pristopa, medtem, ko je velika verjetnost, da bo izbira agilnega pristopa manjšim in srednje velikim projektom predstavljala najboljšo strategijo optimizacije procesa (Sommerville 2010, 706).

Slika 5.2 prikazuje spekter različnih pristopov na kateri so hekerji postavljeni na levo in podrobnih mejnikov na desno (Abrahamsson in dr. 2017, 14).

Slika 5.2 spekter pristopov po Boehm



Vir: (Abrahamsson in dr. 2017, 14)

Ena metodologija ne more ustrezati celotnemu spektru različnih projektov. Vodstvo projekta mora identificirati naravo projekta in nato izbrati primerno razvojno metodologijo (Abrahamsson in dr. 2017, 14).

Cockburn temeljna pravila agilnega razvoja definira kot lahka vendar učinkovita z uporabo človeško in komunikacijsko usmerjenih pravil. Predlagal je naslednje prijeme katerih uporaba povečuje uspešnost projekta;

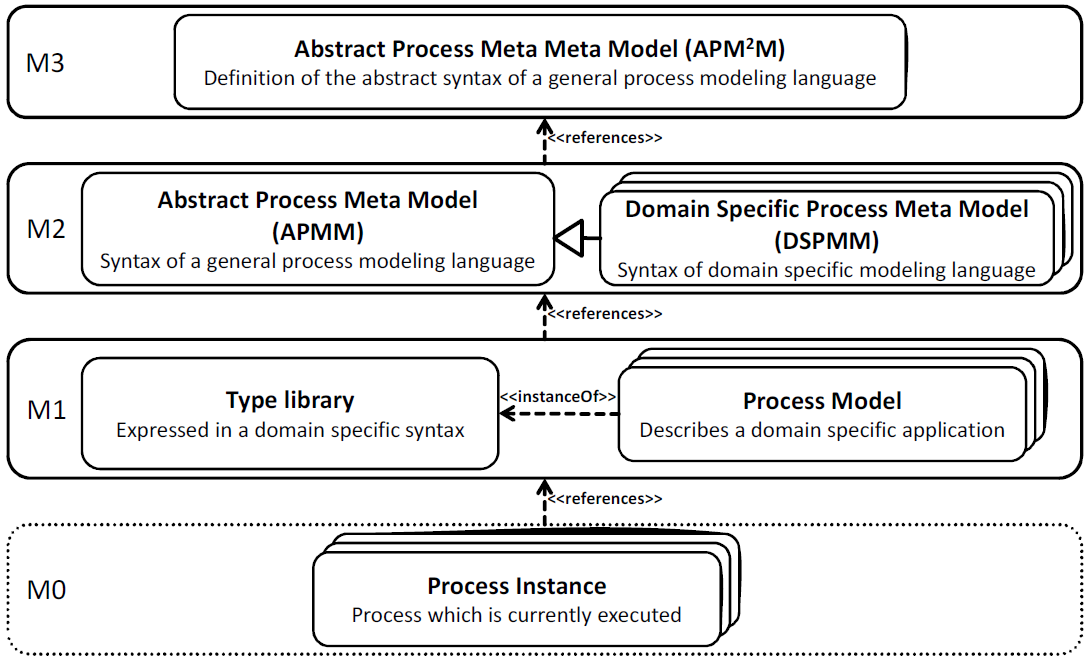
1. Dva do osem razvijalcev v eni sobi,
2. uporabniki na lokaciji,
3. kratki inkrementi,
4. avtomatizirani regresijski testi,
5. izkušeni razvijalci (Abrahamsson in dr. 2017, 15).

Miller predlaga naslednje karakteristike, ki omogočajo skrajšanje življenjskega cikla projekta;

1. Modularnost na stopnji razvoja,
2. iteracije s kratkimi cikli za omogočanje hitre verifikacije in popravkov,
3. časovno vezani iteracijski cikli od 1 do 6 tednov,
4. varčevanje z namenom odstranjevanja nepotrebnih aktivnosti,
5. prilagodljivost na morebitna tveganja,
6. inkrementalni pristop, ki omogoča apliciranje funkcionalnosti v kratkih korakih,
7. konvergenten pristop za zmanjšanje tveganj,
8. naklonjenost ljudem nad tehnologijo,
9. delo na način sodelovanja in komuniciranja (Abrahamsson in dr. 2017, 16).

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). Za potrebe tega bomo uporabili perspektivno-orientiran[[10]](#footnote-10) proces modeliranja, ki se nanaša na sklad meta modela (sl. 5.X) in služijo kot pomoč pri definiciji konstruktov procesnega modeliranja. Temeljna ideja meta modela je izgradnja elementov modeliranja na abstraktnem nivoju, ki se lahko uporabijo za definiranje konstruktov na fizičnem nivoju (Jablonski 2010, 399).

Slika 5.X sklad meta modela perspektivno-orientiranega procesa modeliranja



Vir: (Jablonski 2010, 400)

Kot prikazuje slika 5.X so procesni modeli skupaj s konstrukti definirani v plasti modeliranja M1. Konstrukti so glede na domeno uporabe definirani v plasti M2. Definirajo jih procesni jeziki na nivoju domenskega[[11]](#footnote-11) in abstraktnega[[12]](#footnote-12) 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. Kot zadnja, plast M0 predstavlja delujočo instanco procesa oziroma procesa v uporabi (Jablonski 2010, 400).

Meta model mora pokrivati tako dimenzijo produkta kot dimenzijo procesa metode inženiringa. Produktna dimenzija mora definirati artefakte, ki se morajo izdelati in njihovo medsebojno povezanost. Procesna dimenzija mora definirati postopke, ki so potrebni za izdelavo artefaktov in njihove izvajalce (Engels in Sauer 2010, 413).

Produktna je definirana s procesnimi modeli procesna dimenzija pa z uporabo transformacij. Te predstavljajo manualne ali avtomatizirane naloge razvojnega procesa. Tako produktni del definiramo z artefaktnim modelom, procesni del pa z modeli poteka dela in modeli opravil.

Vpliv eksekucije nalog se bo modeliralo s transformacijskimi pravili na artefaktnem modelu (Engels in Sauer 2010, 414).

Tekom razvoja discipline nam inženiring metod poda različne modele in jih v svoji domeni označuje za metode inženiringa programske opreme (Engels in Sauer 2010, 412).

1. Institute of Electrical and Electronic Engineers – IEEE. [↑](#footnote-ref-1)
2. SoftwareEngineering – Metamodel for Development Methodologies. [↑](#footnote-ref-2)
3. Software & Systems Process Engineering Meta-Model – SPEM. [↑](#footnote-ref-3)
4. Object Management Group – OMG. [↑](#footnote-ref-4)
5. Meta Object Facility – MOF. [↑](#footnote-ref-5)
6. Unified Modeling Language – UML. [↑](#footnote-ref-6)
7. Inženiring meta-metod. [↑](#footnote-ref-7)
8. Meta-Method for Software Engineering Methods. [↑](#footnote-ref-8)
9. Entity-relationship – ER. [↑](#footnote-ref-9)
10. (angl.) Perspective Oriented Process Modeling – POPM. [↑](#footnote-ref-10)
11. (angl.) Domain Specific Process Meta Model – DSPMM. [↑](#footnote-ref-11)
12. (angl.) Abstract Process Meta Model – AMM. [↑](#footnote-ref-12)