Namena knjige:

* prikazati kako model-based pristup može da poboljša svakodnevnu praksu softverskih programera
* cilj knjige je poveća efikasnost i efektivnost u procesu razvoja softvera
* razumeti kada, gde i kako uvesti MDSE principe

**1. Introduction**

* apstrakcija – kognitivan proces
* apstrakcija – mogućnost pronalaska zajedničkih osobina prilikom raznih posmatranja, i na osnovu toga izvođenje generalizovane reprezentacije
* apstrakcija u oblasti nauke i tehnologije – posmatra se kao modelovanje
* model – pojednostavljena slika realnosti, definisana kako bi se izvršio neki zadatak ili dostigao neki dogovor
* model ne opisuje realnost u potpunosti, nego opisuje samo deo realnog sveta, deo od interesa
* model je apstrakcija, aproksimacija realnog sistema

**Svrha i upotreba modela**

* modeli se koriste u raznim industrijama pored razvoja softvera – hemija (model atoma), fizika, matematika, građevina
* modeli – pružaju mogućnost za razumevanje osnova koncepata; za dalje učenje i izučavanje kompleksnijih koncepata/procesa/teorija
* modeli – za opisivanje ponašanja, elemenata, za davanje budućih predikcija
* **modeli** poseduju bar dve **uloge**; primenjujući proces apstrakcije realnog sveta:
  + redukcija – modeli prikazuju samo relevantan (delovi od interesa) deo realnog sveta i mogućnosti u okviru njega
  + mapiranje – modeli predstavljaju abstrakciju i generalizaciju dela realnog sveta koji modelujemo
* potrebno je da model ima svrhu
* **svrha modela** može biti:
  + deskriptivna – za opisivanje realnog sistema, koriste se za razumevanje i komuniciranje (nakon implementacije; npr. class dijagram nakon završene implementacije projekta)
  + preskriptivna – za definisanje opsega i detalja na kom će se izučavati neki problem; modeli koršćeni za implementaciju, sama implementacija modela [definisanje preciznih pravila i uputstava o tome kako nešto uraditi] (pre implmenetacije; npr. class dijagram pomoću kojeg se generiše kod)
* modelovanje omogućava i sintaksnu analizu, proveru modela, simulaciju modela, transformacije modela i izvršavanje modela
* model uvek postoji, samo je pitanje da li je model eksplicitan ili se radi o zamišljenom modelu (koji postoji u samo glavi dizajnera)

**Modelovanje u procesu razvoja softvera**

* Model-Driven Software Engineering prakse za povećanje efikasnosti i efektivnosti u procesu razvoja softvera
* inženjerstvo vođeno modelom (MDE) – npr. građevina; 1. model pa gradnja
* modeli – definisani na različitim nivoima apstrakcije
* modeli – ponovna iskoristivost, koja dovodi do povećanja produktivnosti a istovremeno omogućava održavanje kvaliteta i performanse
* Martin Folerova ***klasifikacija modela***:
  + modeli kao skice – modelovanje radi modelovanja; za komuniciranje ideje (npr. class dijagram nakon završenog projekta)
  + modeli kao nacrti – modeli se koriste da definišu kompletnu i detaljnu specifikaciju sistema; instrukcije pre implementacije (npr. class dijagram pre razvoja)
  + modeli kao program – aplikacija se generiše automatski na osnovu modela; model se koriste umesto koda za generisanje aplikacije (generator koda); modeli koji su u potpunosti izvršivi
* tokom procesa razvoja softvera, koriste se modeli na različite načine. Npr. u procesu dizajna, koriste se modeli kao skice za potrebe diskusija, zatim u sklopu definisanja kompletnih modela, mogu se koristiti modeli kao nacrti, a oni se mogu dalje preraditi tako da se dobije sistem koji koristi generator koda kako bi minimizovao proces pisanja koda
* svrha modela, zavisi od potreba i namene modela – ide od alata za komunikaciju do izvršivog artefakta/generatora koda

**2. MDSE Principles**

**MDSE osnove**

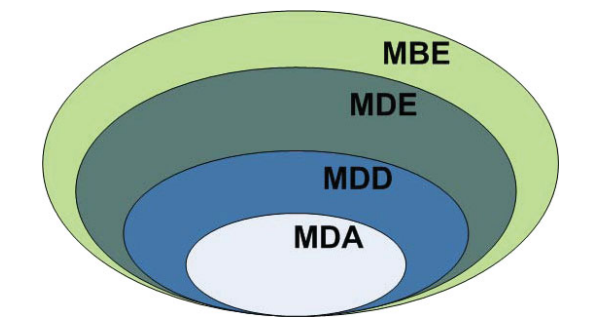
* MDSE može biti definisan kao metodologija za primenjivanje prednosti modelovanja u procesu razvoja softvera. Generalno metodologija se sastoji iz sledećih aspekata:
  + koncepata – komponente koje grade metodologiju
  + notacija – način na koji su koncepti predstavljeni
  + procesa i pravila – aktivnosti (i pravila) koje vode do realizacije krajnjeg proizvoda
  + alata – aplikacije koje olakšavaju izvršavanje produkcionih procesa
* u kontekstu MDSE – glavni koncepti su model i transformacije

modeli + transformacije = softver

* MDSE:
  + koncepti: modeli, transformacije(manipulacije nad modelima)
  + notacija: jezik za modelovanje
  + procesi i pravila: za definisanje nivoa apstrakcije modela, …
  + alati: IDE koje omogućava definisanje modela i transformacija
* MDSE vrlo ozbiljno shvata izraz “sve je model”
* "sve je model" - mogu se i transformacije i jezik za modelovanje posmatrati kao modeli (transformacije kao modeli operacija koje se izvršavaju nad modelima; jezik za metamodelovanje kao alat za modelovanje modela)
* Metamodelovanje -> modelovanje modela ~ modelovanje jezika za modelovanje (modelovanje jezika za opisivanje modela)
* Meta-metamodelovanje -> modelovanje metamodela (za opisivanje metamodela)
* modelovanje VS crtanje
  + Crtanje je proces kreiranja slike, koja potencijalno podlaže nekim sintaksnim pravilima, dok sa druge strane, modelovanje predstavlja mnogo kompleksniju aktivnost, gde postoji definisana semantika koja omogućava preciznu razmenu informacija.
  + modelovanje donosi i mogućnosti sintaksne analize, provere modela, simulacije modela, transformacije modela i izvršavanje modela

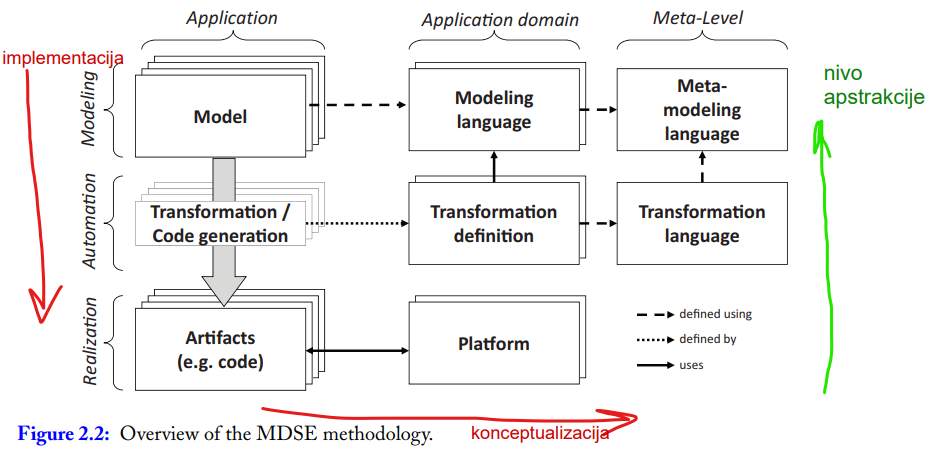
**MD\* akronimi**

* MDA – je vizija MDD-a od strane OMG grupe, zasniva se na određenim OMG strandardima.
* MDD – paradigma koja koristi modele kao primarne artefakte u procesu razvoja.
  + u MDD, softver se dobija kroz nekoliko serija M2M transformacija + potencijalno M2T koja reprodukuje krajnji kod.
* MDE - superset MDD; kako E u okviru MDE nagoveštava, MDE ide izvan čistog procesa razvoja i sastoji se iz drugih model-based zadataka celokupnog procesa softerskog inženjeringa
  + MDE je iterativan i inkrementalan proces.
* MBE - "model-based engineering" ili "model-based development", smatra se blažom verzijom model-driven engineering (MDE).
  + u MBE modeli igraju bitnu ulogu, ali nisu ključni artefakti u procesu razvoja. Razvoj nije vođen modelima (models do NOT “drive” the process as in MDE).
* Svi model-driven procesi su model-based, ali obrnuto ne mora biti tačno.



**Pregled MDSE metodologije**

* narednih 5 naslova se bavi:
  + razjašnjavanjem šta znači da modelovanje može biti primenjeno na različitim nivoima apstrakcije
  + opis pristupa modelovanja
  + uloga i priroda transformacija
* MDSE nudi celokupan pogled na razvoj sistema
* modelovanje na različitim nivoima apstrakcije -> posmatranje s obzirom na dve dimenzije – konceptualizacija i implementacija



* dimenzija implementacije – bavi se *mapiranje* modela na postojeći ili budući sistem, postoje:
  + nivo modelovanja – nivo gde se definišu modeli
  + nivo realizacije – nivo gde se implementiraju rešenja kroz artefakte
  + nivo automatizacije – nivo gde se mapira iz nivoa modelovanja u nivo realizacije
* dimenzija konceptualizacije – orijentisano prema definisanju konceptualnih modela za opisivanje realnosti, postoje:
  + nivo aplikacije – modeli aplikacije su definisani, pravila transformacija su izvršena i generisane su izvšive komponente
  + nivo aplikativnog domena – nivo gde su definisani jezik za modelovanje, transformacije i platforma za implementaciju za specifični domen
  + meta-nivo – definisani koncepti modela i transformacija
* MDSE flow – od modela do izvršivih artefakta kroz niz transformacija
  + ponovna iskoristivnost modela, izvršavanje na različitim platformama
  + modeli su definisani s obzirom na jezik za modelovanje, odnosno s obzirom na jezik za metamodelovanje
  + transformacije su definisane na osnovu pravila za transformacije, koja su definisana određenim jezikom za specifikaciju transformacija

**MDSE; domeni, platforma, tehnički prostor i scenarija**

* prostor/domen problema – definisanje šta je problem
  + oblast koja treba biti istražena kako bi se rešio problem
  + domenski model – konceptualni model domena problema
    - opisuje entitete, atribute, uloge, veze i interakcije iz domena problema
    - namena: razumeti osnovne koncepte domena problema
    - definisan je bez dizajnerskih i implementacionih detalja
* prostor/domen rešenja
  + prikupljanje zahteva – šta je očekivani ishod
  + dizajniranje – kako dostići željeni ishod
* tehnički prostor – definiše specifikaciju, implementaciju i deplojment aplikacije
  + odlučivanje o alatima i formatima koji će se koristiti za modele, transformacije i implementaciju
  + prostire se i kroz prostor problema i kroz prostor rešenja
  + primeri tehničkih prostora: MDE, Java, XML (svi su na različitim nivoima apstrakcije, zavisno od toga kroz koje prostore se prožima)
    - MDE se porstire kroz ceo domen problema i domen rešenja, dok se Java prostire samo kroz deo implementacije rešenja
  + tokom procesa razvoja softvera, moguće je kretati se iz jednog tehničkog prostora u drugi, koristeći injektore(T2M) i ekstraktore(M2T) = transformacije

**Jezici za modelovanje**

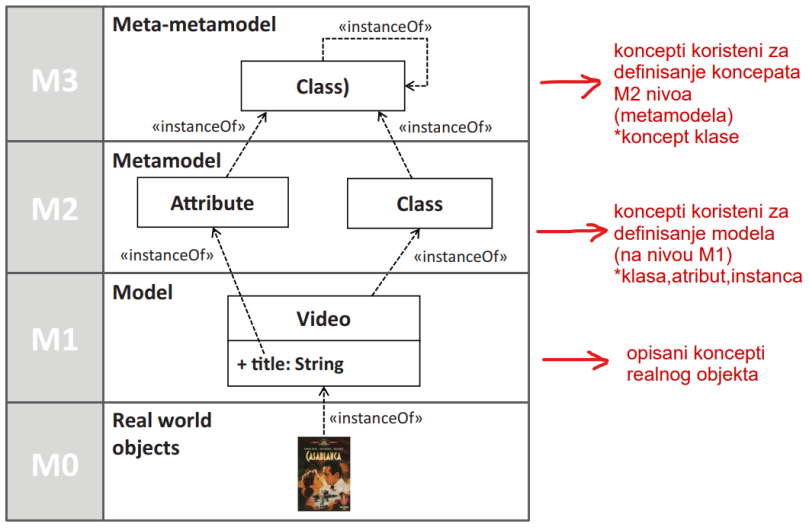
* dve klase jezika za modelovanje:
* DSL (Domain-Specific Language)
  + jezici dizajnirani za specifični domen, (za potrebe određenog domena), da olakšaju zadatke ljudi iz tog domena
  + da unaprede poslovanje nekog domena, za lakše rukovanje nekim domenom, usko specijalizovani
  + DSL ne mora da obavlja sve matematičke operacije
  + primeri: HTML (za opis i izgled web stranica), CSS, VHDL (za opis hardverskih komponenti), SQL (za rad sa podacima, pristup bp), BPMN (za opis poslovnih procesa) …
* GPL (General-Purpose Language)
  + jezik opšte namene
  + može se primeniti na bilo koji domen
  + mora da obalja sve matematičke operacije (Turing complete)
  + primeri: XML, UML, Petrijeve mreže
* C, C++, Java – to su GPL

**Metamodelovanje**

* model je apstrakcija realnog sveta, a metamodel je apstrakcija modela
* metamodeli – modeli koji opisuju modele (metamodeling jezik – za definisanje modela)
* meta-metamodeli – modeli koji opisuju metamodele (meta-metamodeling jezik – za definisanje metamodela)
* metamodeli – jedna od primena: za definisanje novog jezika za modelovanje

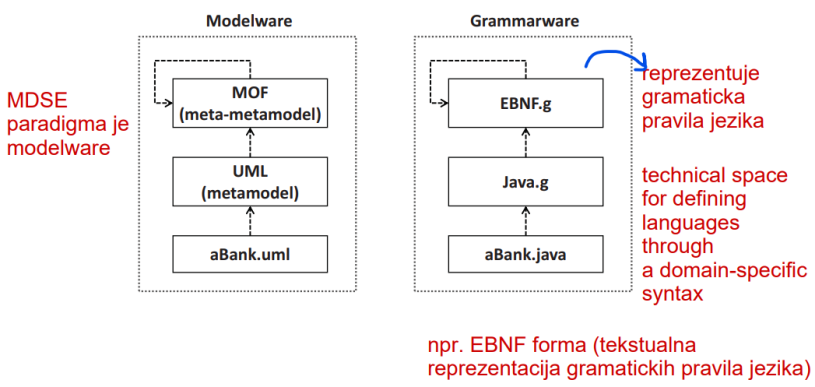
**MOF konceptualni nivoi/ nivoi apstrakcije**

* M0 – realni svet; objekti realnog sveta, nema nikavog modelovanja (konkretna instanca modela ???)
  + film; student
* M1 – model, pojednostavljena predstava realnog sveta
  + koncept Video sa atributima; konkretan tip entiteta Student sa atributima
* M2 – metamodel, opisuje koncepte za definisanje modela (M1 nivo)
  + Film: atribut, klasa, instanca
  + Student: definišemo šta je TE, TP, veze, rekurzija …
  + M2 nivo predstavlja apstraktnu sintaksu jezika, definiše sve koncepte, veze između koncepata i ograničenja; apstraktnom sintaksom nismo definisali predstavu koncepata, nije konkretna sintaksa
    - konkretna sintaksa je kad rombu pridružimo TP
  + **metamodel** – opis koncepte jezika, njihove veze i ograničenja + opis predstave (simbolička predstava)
* M3 – meta-metamodel, za definisanje koncepata iz M2 nivoa
  + podloga svakog jezika
  + kako se bilo šta može predstaviti
  + uvodi pojam klase i asocijacije, za opis bilo čega iz realnog sveta
* na bilo kom nivou kažemo da model “odgovara” svom metamodelu – model conforms to its metamodel



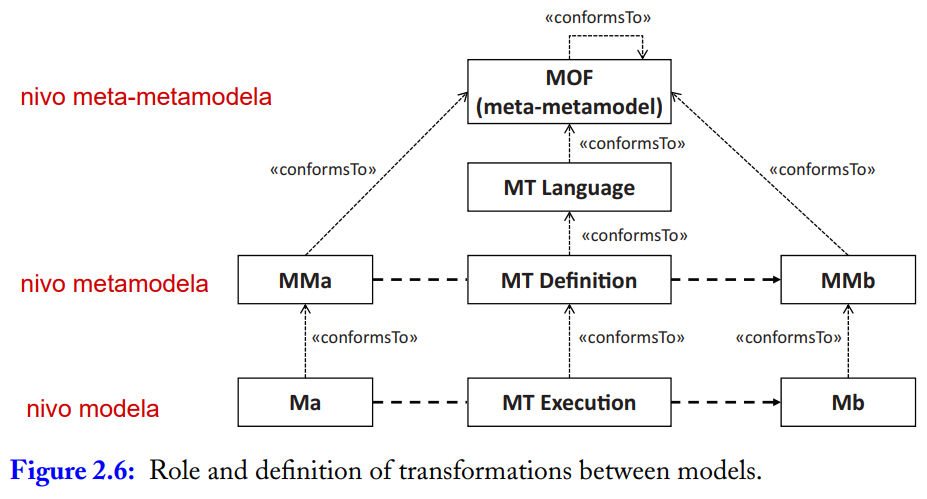
**Modelware VS grammarware**

* modelware i grammarware su različiti tehnički prostori
* grammarware – definisanje jezika na osnovu **EBNF** forme
* EBNF je tekstualna reprezentacija gramatičkih pravila jezika (EBNF – jezik svih jezika)
* modelware – definisanje jezika na osnovu **MOF**-a



**Transformacije**

* transformacije se definišu na nivou metamodela, a izvršavaju se na nivou modela (koji se podudaraju/u skladu su sa svojim metamodelom)
* transformacije se izvršavaju između source i target modela (Ma i Mb)
* MDSE nudi jezik za definisanje transformacija modela



* primer transformacije iz EER u relacioni model:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M3 | MOF | specifikacija  transformacije  modela  → | M3 | MOF |
| M2 | EER (Metamodel + konkretna sintaksa) | M2 | Relacioni metamodel + konkretna sintaksa |
| M1 | model IS FTN | M1 | model IS FTN |
| M0 | FTN | M0 | FTN |

**Klasifikacija modela**

* postoji više vrsta klasifikacije modela, zavisno od toga na osnovu čega klasifikujemo
* *klasifikacije* prema nivou apstrakcije:
  + modeli koji opisuju zahteve na vrlo visokom nivou apstrakcije (bez implementacionih detalja)
  + modeli koji opisuju ponašanje sistema u smislu algoritama i načina čuvanja podataka (bez tehničkih i tehnoloških detalja)
  + modeli koji opisuju sve tehničke detalje
* *klasifikacija* modela na osnovu vrste ponašanja:
  + statički – fokus na statičkim aspektima sistema – podaci kojima se upravlja, struktura i arhitektura sistema
    - opis osnovnih koncepata i njihovih veza
  + dinamički – fokus na dinamičkom ponašanju sistema – izvšavanje aktivnosti i algoritama, kolaboracija između komponenti sistema
    - opis ponašanja koncepata u smislu aktivnosti, događaja i interakcija koje se dešavaju između njih
* separacijom se uviđaju različiti pogledi na sistem, ali uvek treba posmatrati sistem kao celinu i gledati i statičke i dinamičke aspekte; svaki model se fokusira na različitu perspektivu, ali krajnji cilj je ima kompletnu sliku sistema, iako se aspetki odvojeno posmatraju
* npr. UML – dozvoljava dizajneru da prikaže neki aspekat na razne načine, koristeći različite tipove dijagrama

**Prihvatanje MDSE u industriji**

* učenje novog alata i novih tehnika inicijalno smanjuje produktivnost programera i kvalitet proizvoda i to je razlog zašto ljudi ne vole i izbegavaju promene. Nakon prevazilaženja inicijalnih problema, uviđaju se prednosti.

**Alati – alati za crtanje VS alati za modelovanje**

* Alati istovremeno mogu biti i za crtanje i modelovanje ali najčešće to nije slučaj.
* Postoje alati za modelovanje koji koriste tekstualnu konkretnu sintaksu za specifikaciju modela i stoga nemaju podršku za crtanje modela.
* Alati za tekstualnu sintaksu: XText, TCS, EMFText, TextX
* Mnogi alati za crtanje nisu alati za modelovanje. Alat za crtanje je istovremeno i alat za modelovanje ukoliko alat razume sliku. Potrebno je da alat može bar da validira model, da validira da je model ispravna instanca odgovarajućeg metamodela. Alat za crtanje dozvoljava dizajneru da nacrta nešto što sematnički nije tačno.
* Preporuka je da se uvek koriste alati za modelovanje prilikom definisanja modela.
* Alati za modelovanje omogućavaju eksportovanje modela, zadovoljavaju minimalni nivo semantike i kvaliteta modela i omogućavaju rad sa transformacijama modela.

**Eclipse i EMF**

* platforma koja je postala istaknuta u MDSE svetu je Eclipse razvojno okruženje
* EMF (Eclipse Modeling Framework) – tehnologija u Eclipse-u za model-driven inženjerstvo
* EMF omogućava definisanje metamodela na osnovu jezika za metamodelovanje – Ecore
* Ecore – implementacija MOF-a u Eclipse-u
* EMF nudi i editor za razvoj modela, omogućava (de)serijalizaciju modela u XMI format …

**Kritike MDSE**

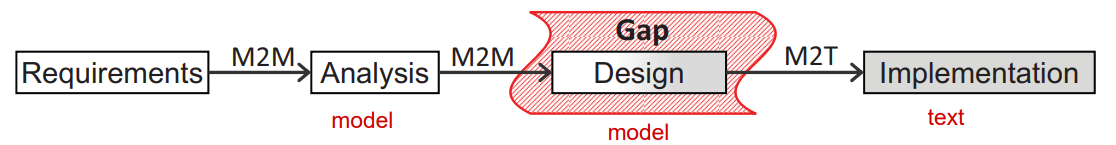
* modeli su pojednostavljena reprezentacija realnog sveta – postoje kritike jer je model pojednostavljenje
* svaka praksa u razvoju softvera treba da poveća produktivnost, a u MDSE se mora naučiti dobro definisanje modela, definisanje transformacija, a za to je potrebno vreme …
* dijagrami se smatraju samo lepim slikama; često su nejasni i nedovoljno definisani
* ljudi često ne vide celokupnu sliku tokom razvoja i zato se može desiti da određenim ljudima pisanje(definisanje) modela samo otežava rad

**3. MDSE Use Case**

* scenariji korišćenja MDSE:

1. **Automatizacija razvoja softvera**

* od zahteva sistema, različitim M2M i M2T transformacijama (kroz nekoliko faza) dobijamo izvršivu aplikaciju
* u svakoj fazi, modeli se dobijaju (polu)automatski M2M transformacijama



* model-driven pristup umanjuje razliku/rupu između **zahteva sistema** i **implementacije**
* modeli omogućavaju “povezivanje” potreba(zahteva) poslovanja i realizacije (implementaciju) tih zahteva
* prednosti uvođenja MDSE u razvoj softvera, modeli omogućavaju:
  + olakšanu komunikaciju unutar tima
  + detaljnu dokumentacija
  + ponovnu iskoristivost delova sistema
* da bi se generisao izvršivi sistem na osnovu modela, model mora biti izvršiv
  + npr. imamo definisan class dijagram i na osnovu njega neki “glup” generator koda može da generiše samo skelet java klase (samo su definisani atributi, geteri i seteri), dok neki “pametniji” generator koda može da implementira i CRUD operacija i ponašanje, da kreira stranice i forme …
* jedna od najpoznatijih porodica izvršivih modela – izvršivi UML
* Na koji način se izvršivi modeli izvršavaju? Koje strategije postoje?
  + Code-generation i model-interpretation
  + To su strategije za implementaciju izvršivih alata, na osnovu čega se izvršivi modeli zapravo izvršavaju

**Generator koda**

* generatori koda generišu izvršivi kod aplikacija na osnovu modela
* izvršivi kod može biti u programskom jeziku po želji
* kompajleri proizvode izvršive binarne fajlove na osnovu izvornog koda, na isti način code-generatori generišu izvršivi kod na osnovu modela
* cilj je minimizovati količinu generisanog koda koji ostvaruje željenu funkcionalnost

**Parcijalno generisanje koda VS Potpuno generisanje koda**

* ukoliko je ulazni model nepotpun ili generator koda nije dovoljno pametan da izgeneriše informacije koje fale, može se kreirati parcijalna implementacija sistema
* treba biti obazriv, jer to znači da programeri ručno treba da dorade ostatak koda
* potrebno je da postoje zaštićene zone, u kojima se nalazi samo ručno pisani kod
* fokusirati se na potpuno generisanje delova sistema nasuprot parcijalnom generisanju celog sistema, time se umanjuje rizik potencijalne greške !!!

**Prednosti generatora koda**

* model se može ponovo iskoristiti ili nadograditi
* generisana implementacija je model- i proces- agnostic (može biti primenjena u različitim okruženjima)
* dozvoljava klijentima da biraju izvršno okruženje
* jednostavno za održavanje i debagovanje jer se generator koda sastoji iz rule-based pravila

**Nedostaci generatora koda**

* može se desiti da je generisan kod nepoznat developerima, zato je bitno da generator koda prolazi Turingov test (test u kome se poredi kod koji je izgenerisan od strane čoveka i kod koji je izgenerisan od strane generator koda, ako čovek ne može zasigurno da odredi koji kod je napisan od strane čoveka, generator koda je prošao test)

**Interpretacija modela**

* kod interpretacija modela, ne vrši se generisanje koda na osnovu modela
* model interpreter implementira generic engine koji parsira i izvršava model “on the fly”
* brži od generator koda jer samo interpretira kod
* dozvoljene promene modela u toku izvršavanja aplikacije (samo se vrši parsiranje nove verzije modela)
* sa code-generation pristupom dobijamo izvršivi kod koji treba negde deplojovati, a sa model interpreterom dobijamo odmah izvršivi model/aplikaciju (nije potrebno deplojovanje)
* deplojovanje nije potrebno jer je model izvršiva verzija aplikacija, ali baš zato ne postoji izvorni kod; s obzirom da je sam model izvršiva aplikacija, na neki način “model jeste kod”, jer ne postoji drugi kod
* viši nivo apstrakcije – platforma mora da se sastoji iz izvršivog engine-a koji može da parsira i izvršava model
* Nedostaci:
  + klijenti moraju da instaliraju kod sebe platformu sa interpreterom
  + izvorni kod nije dostupan

**Kombinovanje generatora koda i interpretacije modela**

* postoji hibridni pristup, koji kombinuje code-generation i model-interpretation
* ne postoji odgovor koji je bolji ili lošiji pristup
* odabir pristupa zavisi od ekspertize tima, potreba i zahteva aplikacije, faze razvoja …

1. **Interoperabilnost (kompatibilnost) sistema**

* interoperabilnost/kompatibilnost – mogućnost 2 ili više sistema/komponente da razmenjuju informacije i koriste te razmenjene informacije
* interoperabilnost je potrebna za:
  + forward i reverse engeneering
  + sistemsku integraciju …
* interoperabilnost zahteva prevazilaženje sintaksnih i semantičkih razlika između sistema/komponenti
* ideja – obezbeđivanje generičkog seta alata za premoštavanje razlika između komponenti 2 ili više sistema, prilikom čega treba obezbediti:
  + interoperabilnost na nivou podataka – razmena podataka
  + interoperabilnost na nivou operacija – razmena ponašanja
* model-driven interoperability (**MDI**) – pristup za premoštavanje razlika između 2 ili više sistema primenom model-driven tehnika kako bi se omogućila njihova interoperabilnost/kompatibilnost
  + definisanje metamodela za svaki sistem
  + poređenje metamodela, poklapanje zajedničkih koncepata
  + primena M2M transformacija za eksportovanje podataka (modela) kreiranih u okviru prvog sistema u podatke (model) drugog sistema
  + [ 1. T2M (injekcija), 2. M2M transformacije, 3. M2T (ekstrakcija) ]
* osnovni princip MDI je dekompozicija na sintaksno i semantičko mapiranje

**Sintaksno mapiranje**

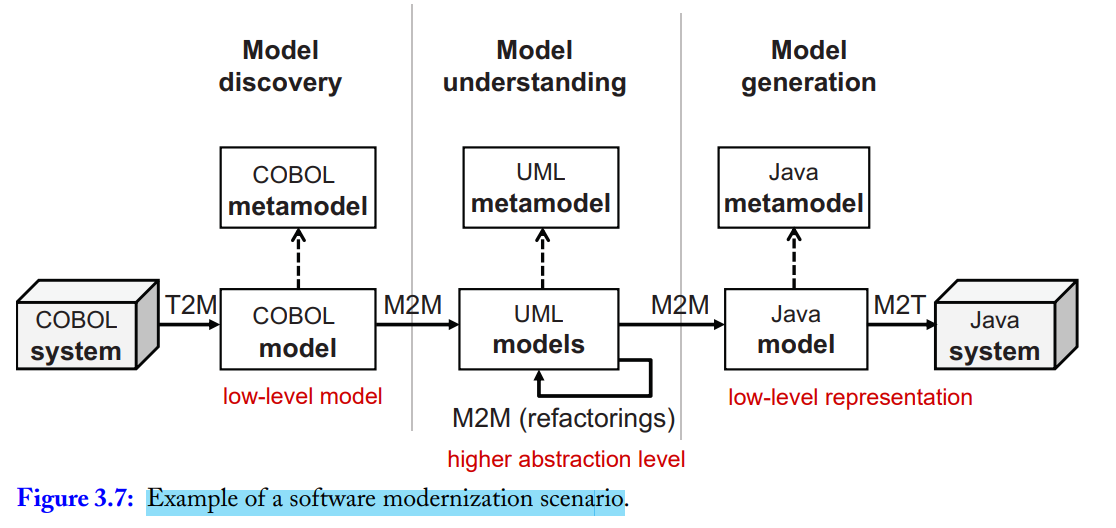
* za prenos između različitih tehničkih prostora
* ideja – koristiti projektore
* projektori:
  + injektori (T2M) – omogućavaju injektovanje sadržaja ulaznih fajlova i kreiranje modela (transformacija ulaznog fajla u model)
  + ekstraktori (M2T) – omogućavaju ekstraktovanje sadržaja modela kao izlazne fajlova (generisanje izlaznog fajla na osnovu modela)

**Semantičko mapiranje**

* poravnavanje kocepata iz domena oba sistema
* implementirano kao M2M transformacije

1. **Reverse engineering**

* Reverse engineering – proces kreiranja reprezentacije (visokog nivoa) legacy sistema
* [sistem je razvijen nekom zastarelom tehonologijom, ali želimo da vizualizujemo i razumemo arhitekturu sistema, način manipulacije podacima, pravila poslovanja …]
* MDSE kao rešenje nudi **MDRE** – primena model-driven inženjeringa za obezbeđivanje model-based pogleda na legacy sistem
* MDRE koristi artefakte legacy sistema kako bi kreirao set modela koji reprezentuje sistem
* 3 faze u okviru MDRE procesa:
* **Model discovery**
  + prelazak iz heterogenog realnog sveta u homogeni svet modela gde su svi artefakti reprezentovani kao modeli; dobijanje inicijalnih raw modela
  + [kreiranje inicijalnih modela na osnovu postojećih legacy artefakta]
* **Model understaning**
  + transformacija raw modela (dobijenih u fazi model discovery) u druge reprezentacije kojima se lakše rukuje
  + [transformacija inicijalnih modela]
* **Model (re)generation**
  + korišćenje modela iz model understaning faze za generisanje željenog izlaza (koda ili refaktorisane verzije sistema)
* primer migracije i deplojmenta COBOL aplikacije na Java platformu



1. **Model-Driven Architecture (MDA)**

* predlog OMG grupe za primenu MDE praksi za razvoj sistema -> MDA
* Istorija: OMG grupa je predstavila MDA 2001. godine.
* MDA je MDE pristup razvoja sistema standardizovan od strane OMG grupe
* prema mišljenu OMG grupe:
  + modeli treba da budu dobro definisani, kako bi omogućili efektnu komunikaciju i razumevanje sistema
  + specifikacija sistema opisana kao skup modela i transformacija
  + modeli “odgovaraju” svojim metamodelima

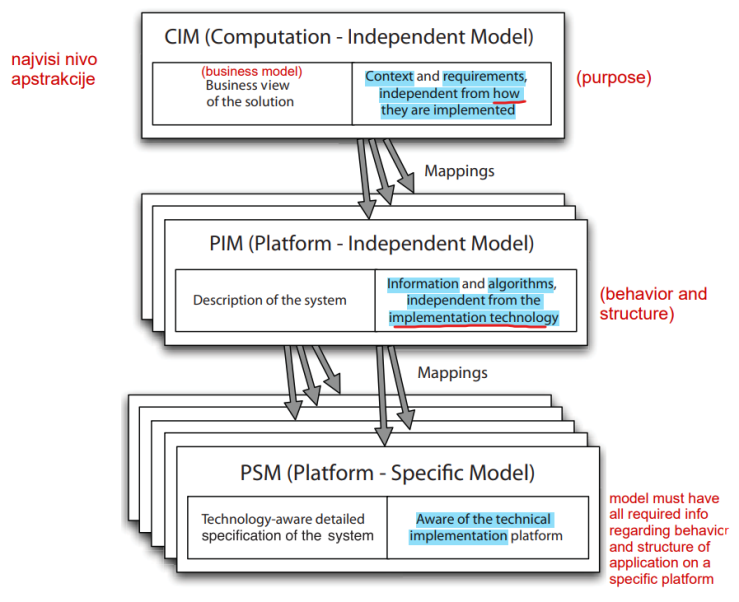
**MDA definicije i pretpostavke**

Glavni elementi vezani za MDA:

* sistem
* prostor/domen problema
* prostor rešenja
* model
* arhitektura
* platforma
* viewpoint   
  (tačka gledišta)
* view (pogled)
* transformacija

**Nivoi modelovanja: CIM, PIM, PSM**

* u okviru MDA postoje 3 nivoa apstrakcije:
* **CIM** (Computation-Independent Model)
  + nivo modelovanja na najvišem nivou apstrakcije
  + reprezentovanje konteksta, zahteva, namene rešenja
  + prikazuje šta se očekuje da rešenje izvršava, ali nezavisno od toga kako je sistem implementiran
  + posmatranje sistema iz biznis perspektive, još se zove i business model
* **PIM** (Platform-Independent Model)
  + opisuje ponašanje i strukturu aplikacije, nezavisno od implementirane platforme
  + generalan i agnostičan
* **PSM** (Platform-Specific Model)
  + opisuje sve neophodne informacije koje se tiču ponašanja i strukture aplikacije na specifičnoj platform
  + uključuje sve tehničke detalje



**Mapiranja**

* Mapiranje se sastoji iz povezivanja/sklapanja različitih metamodela
* npr. ako je PIM nivo opisan Petrijevim mrežama, a PSM nivo opisan UML dijagramom, mapiranje treba da poveze koncepte Petrijevih mreža sa odgovarajućim konceptima UML dijagrama
* mapiranje iz PIM u PSM je definisano kao transformacija definisana između metamodela nezavisnog od platforme i metamodela zavisnog od platforme
* mapiranja između modela ili nivoa modelovanja može biti implementirano kroz transformacije
* cilj je automatizovati mapiranje kroz transformacije