MDSE USE CASES:

* Najbitnije primjena mds-a generalno je automatizacija razvoja softvera (svaka faza, od zahtjeva, do deploya) 🡪 ali ovo je ustvari MDD (MDD je manje od MDSE) MDD kao vidljivi dio mdse-a
* Kako se dobije funkcionalna aplikacija primjenom MDSE-a? Dobije se kroz jednu ili vise transformacija modela, tj. Model to Model transformacija (iz requirement u analysis u design recimo) I na kraju model to text transformacija.

In summary, Model-Driven Development is a specific approach to software development that focuses on using models as the primary artifacts for design and implementation. Model-Driven Software Engineering, on the other hand, is a broader concept that encompasses the use of models throughout the entire software engineering lifecycle, including requirements engineering, design, testing, and maintenance.

Software development automation consists of starting from a high level (or early) representation of

the desired software features and deriving a running application out of it, possibly through a set of

intermediate steps to enable some degree of user interaction with the generation process.

* Krecemo od rane slike (odn. Reprezentacije ) zeljenog softevra I izvodimo funkcionalan sistem iz toga. Kako? Kroz niz medjukoraka, pri cemu je u svakom koraku prisutna interakcija (input) korisnika, tj developera.
* Postoji veliki komunikacijiski jaz izmedju analize zahtjeva I implementacije (ili tacnije problem u komunikaciji izmedju it timova i ljudi iz biznisa, kako ovdje pise) I modelu su se to pokazali kao dobro rjesenje.
* Osim toga, jos neki benefiti uvodjenja modela u razvojni proces su: podspjesuju raspravu izmedju članova tima jer oni hvataju sustinu sistema koji se pravi, pogodni su za dokumentaciju, povecavaju dekompoziciju I modularizaciju sistema, poboljsavaju ponovnu upotrebu dijelova sistema u novim projektima ili u daljem razvoju i odrzavanju postojeceg projekta
* POENTA: najdirektniji benefit mde-a je efektivnija komunikacija između stakeholdera i povecana produktivnost razvojnog tima zbog djelomične automatizacije razvojnog procesa.

\*\*\* kako prevesti execution engine? \*\*\*

* Da bi se generisao funkcionalan inf sistem model mora da bude izvrsiv. Izvrsiv model je onaj model koji je dovoljno kompletan da bi bio izvrsiv. Sa teoretskog aspekta, model je izvrsiv kada je u potpunosti definisana operativna semantika. U praksi izvrsivost ne zavisi samo od samog modela, nego i execution engine-a. (Primjer klas dijagrama. Neki engine ce generisati samo ddl skriptu, a neki funkcionalnu aplikaciju sa podrzanim crudom za svaku klasu, a pocetni model je isti za oba slucaja). Najpoznatija familija izvrsivih modela je izvrsivi UML.

As an example, this advanced generator could assume that for each class in the

diagram the system will need to offer all typical CRUD (create/read/update/delete) operations and

thus, it could directly decide to create all the forms and pages implementing these operations3 (in

fact, studies like [3] show that the CRUD operation account for a staggering 80% of the overall

software functionality in typical data-intensive applications so this advanced code-generator could

save a lot of development time from a minimal system specification).

* Kako bismo omogucili izvrsavanje izvrsivih modela, postoje 2 razliciet alternativne strategije: interpretacija modela I generisanje koda. (code generators = model compilers)

Evo nekih primera izvršnih UML jezika:

1. UML-B: UML-B kombinuje UML i Event-B, formalnu metodu zasnovanu na matematičkom modeliranju događaja. UML-B omogućava formalno modeliranje sistema i njihovu analizu, što rezultira generisanjem izvršnog koda.
2. SysML: SysML (Systems Modeling Language) je proširenje UML-a koje je specifično dizajnirano za modeliranje i analizu sistemskih i softverskih sistema. SysML podržava opisivanje funkcionalnosti, strukture i ponašanja sistema, omogućavajući kreiranje izvršnih modela i simulacija sistema.
3. AADL: AADL (Architecture Analysis and Design Language) jezik je specifično dizajniran za modeliranje i analizu ugrađenih sistema visokog pouzdanja. AADL omogućava detaljno opisivanje hardverske i softverske arhitekture sistema, kao i njihovu analizu performansi i pouzdanosti.

Ovi jezici omogućavaju programerima i inženjerima da koriste UML modele ne samo za dokumentaciju, već i za generisanje izvršnog koda ili simulaciju sistema, čime se povećava produktivnost i kvalitet razvojnog procesa.

Ostali slucajevi koriscenja mdse-a:

Internet stvari (IoT): MDSE se može koristiti za dizajn i razvoj modela za IoT sisteme. Ovi modeli mogu da prikažu interakcije između različitih IoT uređaja, komunikacione protokole i tokove podataka. MDSE može pomoći u automatizaciji generisanja koda za IoT sisteme, olakšavajući razvoj i održavanje složenih IoT aplikacija.

Ugrađeni sistemi: MDSE se može primeniti na razvoj modela za ugrađene sisteme, koji su posvećeni računarski sistemi dizajnirani za specifične zadatke ili funkcije. Koristeći MDSE, programeri mogu da kreiraju apstraktne modele hardverskih i softverskih komponenti ugrađenih sistema i generišu kod na osnovu tih modela, pojednostavljujući proces razvoja i osiguravajući doslednost.

Testiranje i verifikacija softvera: MDSE se može koristiti za modeliranje ponašanja i funkcionalnosti softverskih sistema, koje se zatim mogu koristiti za automatsko testiranje i verifikaciju. Generisanjem test primera iz modela, programeri mogu da se osiguraju da softver ispunjava specificirane zahteve i da se ispravno ponaša u različitim scenarijima.

Održavanje i evolucija softvera: MDSE može pomoći u održavanju i evoluciji softverskih sistema pružajući visoko nivo, apstraktnu reprezentaciju sistema. Korišćenjem modela, programeri mogu da analiziraju uticaj promena, identifikuju zavisnosti i automatizuju propagaciju modifikacija na različite delove sistema.

Chapter 5 koji se naziva **Integration of MDSE in your Development Process** (to je ustvari moja tema, ovo je bilo samo uvod):

* MDSE ne definise koje vrste modela se koriste u kom koraku razvojnog procesa , nak om nivou apstrakcije itd. Organizacija to definise u skladu sa svojim potrebama.
* Integracija MDSE-a se mora odvijati postepeno i u obzir treba uzeti kontekst organizacije.
* 3 aspekta treba uzeti u obzir prilikom usvajanja ove metodologije: organizacioni, menadzerski I socijalni. (pored prostih tehnickih aspekata)

Kako olaksati ovaj process? Neke stvari su očigledne:

* Za pocetak, odabrati manji projekat I projekat koji nije kritican.
* Omoguciti podrsku za potencijalni problem koji može nastati kao I eksternu pomoć

Neke stvari nisu baš očigledne:

* 2 najveca izazova: Nepravilna distribucija napora i nagrada unutar clanova tima I nedostatak socio-tehnicke saglasnosti
* Uvodjem nove metodologije dolazi I do stvaranja novih taskova I uloga. Od tih dodatnih taskova se ocekuje da budu korisni. Osobe u timu koje rade te dodatne taskove nekada ne vide pozitivan uticaj za njih, za druge mozda da. Ako se modeli kreiraju da bi kasnije odrzavanje bilo lakse, osoba vidi pozitivan uticaj pravljenja modela (novog taska) samo ucestvuje u odrzavanju I vidi da je bilo poente to raditi. A obicno grupa koja pravi modele nije grupa koja odrzava pa tu nastaje problem, znaci work distribution da ne bi doslo do problema u motivaciji.

For instance, the organization must be able to match the roles of the target process on

the actual team members, ensuring that each member has the skills required to play that role.

* Recimo, kompacija koja uopste nema iskustva u modelovanju odluci da usvoji MDD sa kompletnim generisanjem koda. Ovo zahtjeva nove uloge, nove zavisnosti izmedju uloga (ko je upoznat sa cim) I volju svih ljudi da nauce nove stvati. Nije trivijalna evolucija uopste.
* Ono sto postoji u tradicionalnom razvoju (tradicionalnim se ovdje smatra razvoj koji ima faze koje slijede) kao faze (nebitno da li je waterfall okvir, scrum okvir itd) jesu analiza zahtjeva, dizajn, implementacija, odrzavanje.. U sustini tu se vec koriste modeli (model based), dizajn model recimo, ali mdse zeli - model driven I zeli automatizaciju tranzicije ovih navedenih faza u razvoju softvera.
* Do sada, nije bas postojala saradnja izmedju mdse I agilnih metodologija. Kritike na model (models don’t run, models cant be tested, are just documentation, require extra work and allignment) te nema agilnosti. To I jeste tacno ako se modeli posmatraju kao skice, ili kao blueprints. Ali tu postoji I momenat izvrsivosti sto je game changer.
* *Modeli su zamisljeni da budu aktivni dio implementacije i verifikacije sistema.* Konstrukcija ce skoro pa biti besplatna ako se dizajn uradi kako treba (dizajn izvrsivih modela).
* \**Agilno Modelovanje*\* kada pricamo o saradnji agile I mdse ima principe: modeluj sa svrhom, Svaki artifact se mora odrzavati pa zato travel light (zrtvuj agilnost radi dostupnosti te informacije? Nije mi najjasnije ovo), Model za svaki aspect pa imaj vise modela, Brz odgovor, Prihvatanje promjene, Primarni cilj je softver koji radi, Inkrementalne primjene, Izvrsive specifikacije imaju prednosti, Dovoljno dokumentacije da bi ljudi koji preuzimaju bili efektivni itd…
* Ne samo da se mdse moze integrisati u agilne metodologije, nego je I obrnuto tacno.
* Domain driven design kaze:

1. focus projekta je sami domen a ne tehnicki detalji,

2. dizajn kompleksnog modela se bazira na modelu.

Mdse kao framework koji omogucava tehnike da bi se DDD ispunio u praksi, mdse kao complement ddd-u.

As such, DDD emphasizes the importance of an appropriate and effective representation of

the problem domain of the system-to-be.To that purpose,DDD provides an extensive set of design

practices and techniques aimed at helping software developers and domain experts to share and

represent with models their knowledge of the domain

* Modeli se mogu koristiti da bi se izveli testovi koje implementacija sistema mora da prodje – **Model Based Testing.** Ti testovi se moraju pretvoriti u izvrsive testove za konkretnu platformu.

Test generation and execution automated.

* Kada se doft kod generise iz modela, nema smisla da se testira kod, nego sami model. **Test Based Modeling**