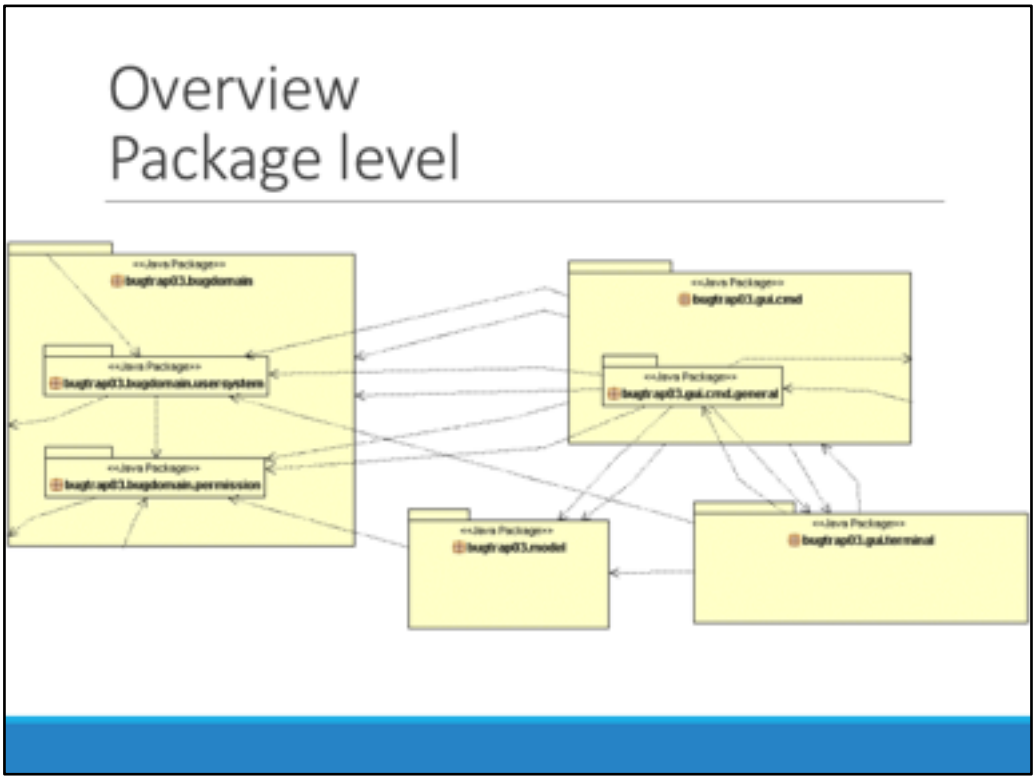
SWOP Iteratie 1
Group 03

M

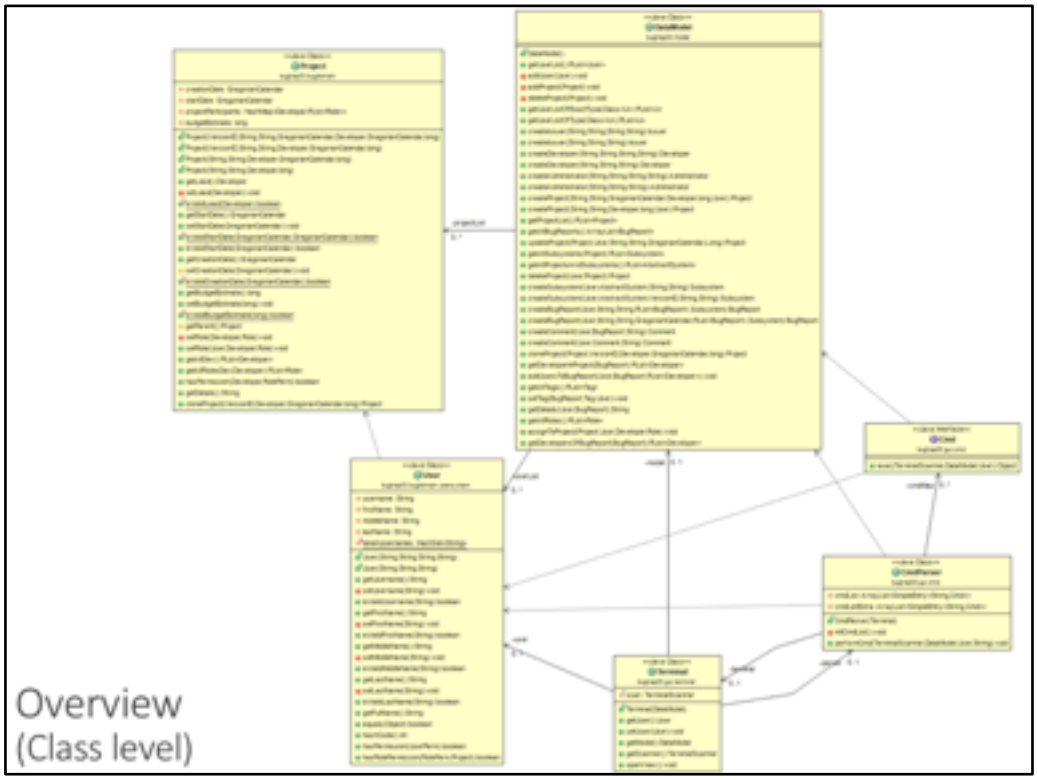


M

Notice: no dependencies from BugDomain to others.

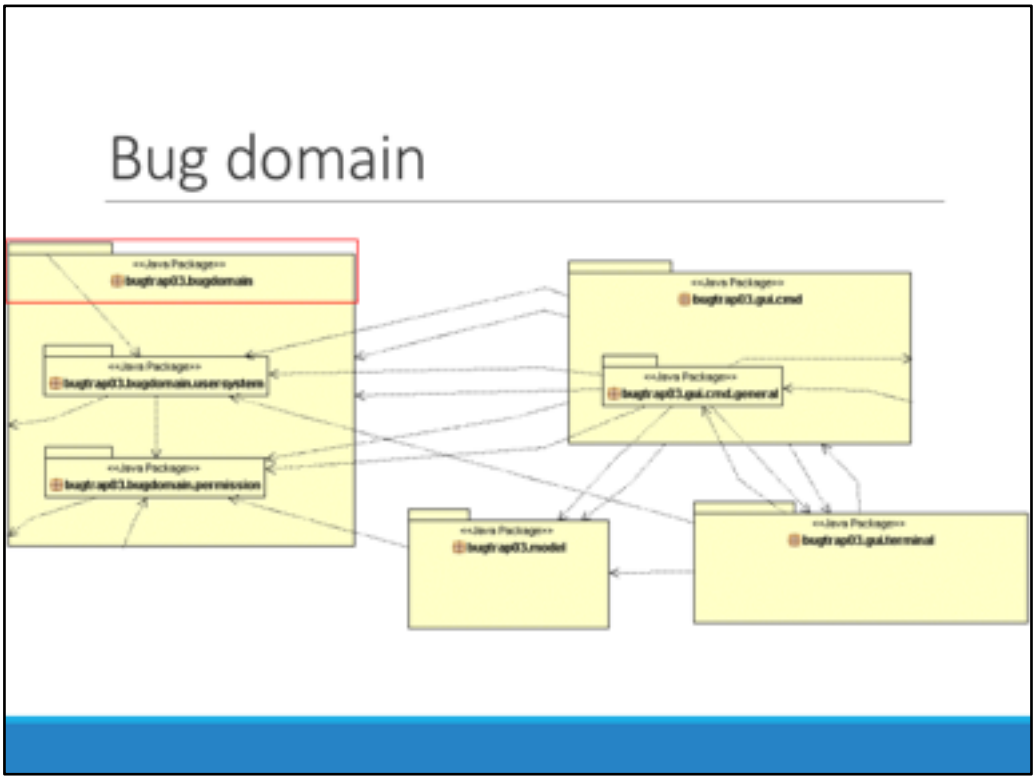
Notice: No dependencies from Model to UI.

=> Low coupling



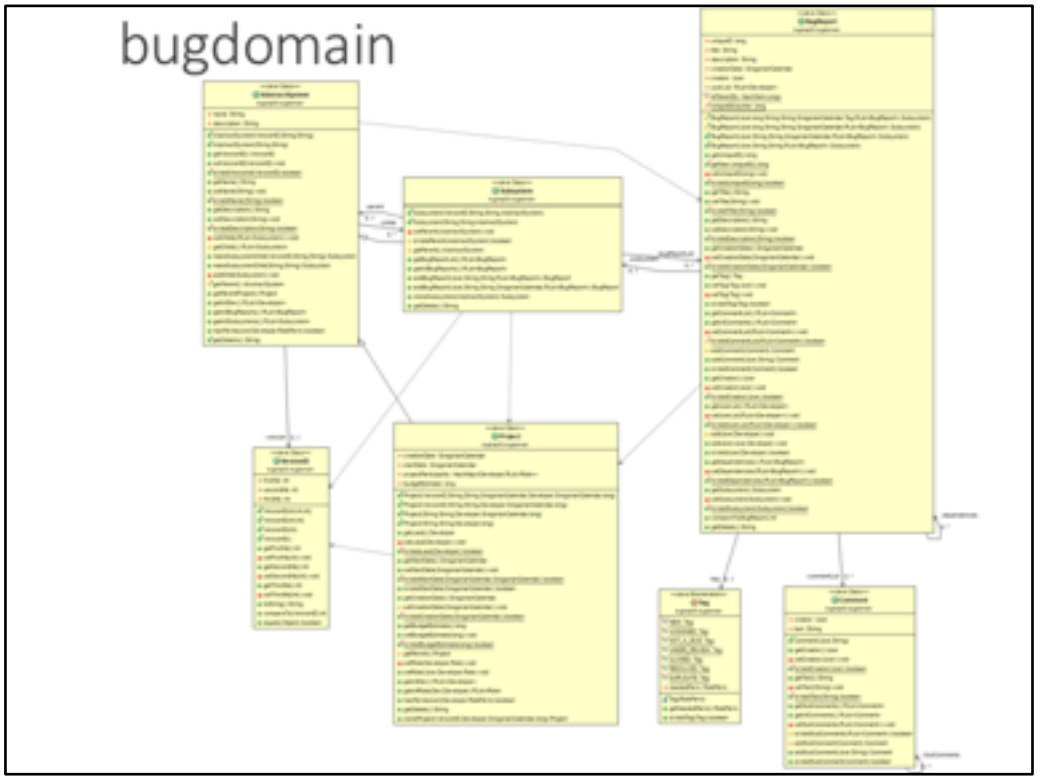
M

=> General overview of the more important classes involved in the interaction between the packages.



K

Next we will have a look at the bugdomain package.



K

Project + Subsystems

* AbstractSystem (name, vID, descip, childs). Proj & Sub inherits.
+ Polymorhispm
+ High Cohesion
- High Coupling

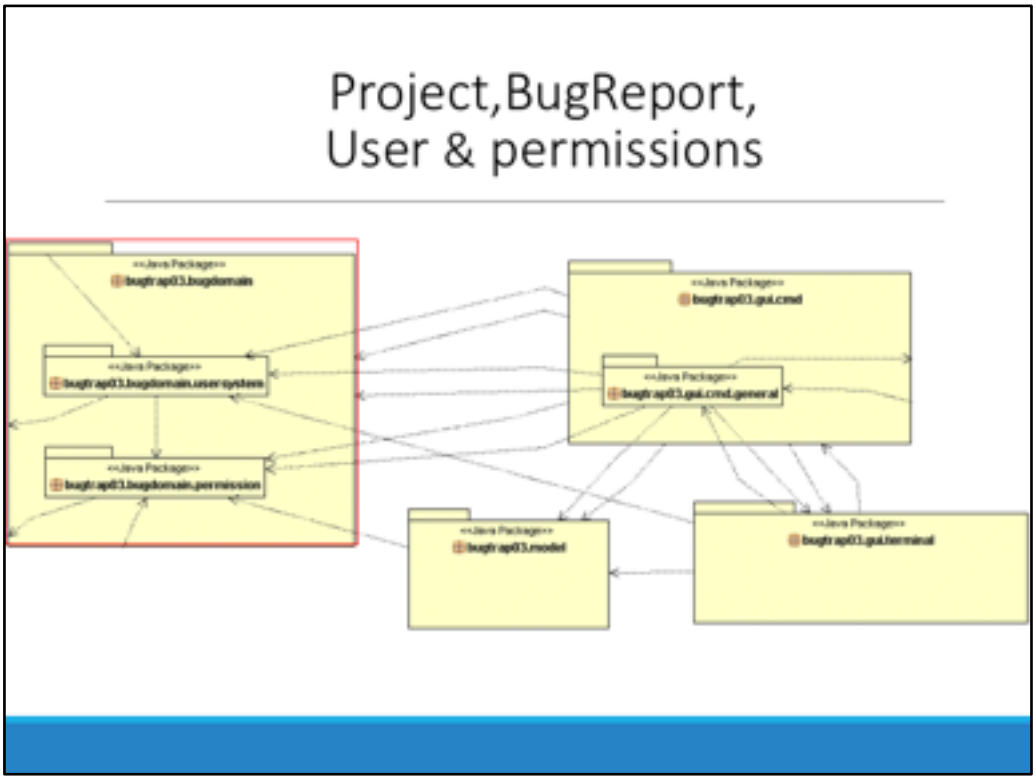
* Project w attribute SubSystem.
- Code redundancy;
- High Coupling (Lower than first)

* Project & Subsystem as one class (Composite)
+ Low Coupling
- Low Cohesion (Can't differentiate between Proj & Sub. Code for Bugreports is redundant for Proj)

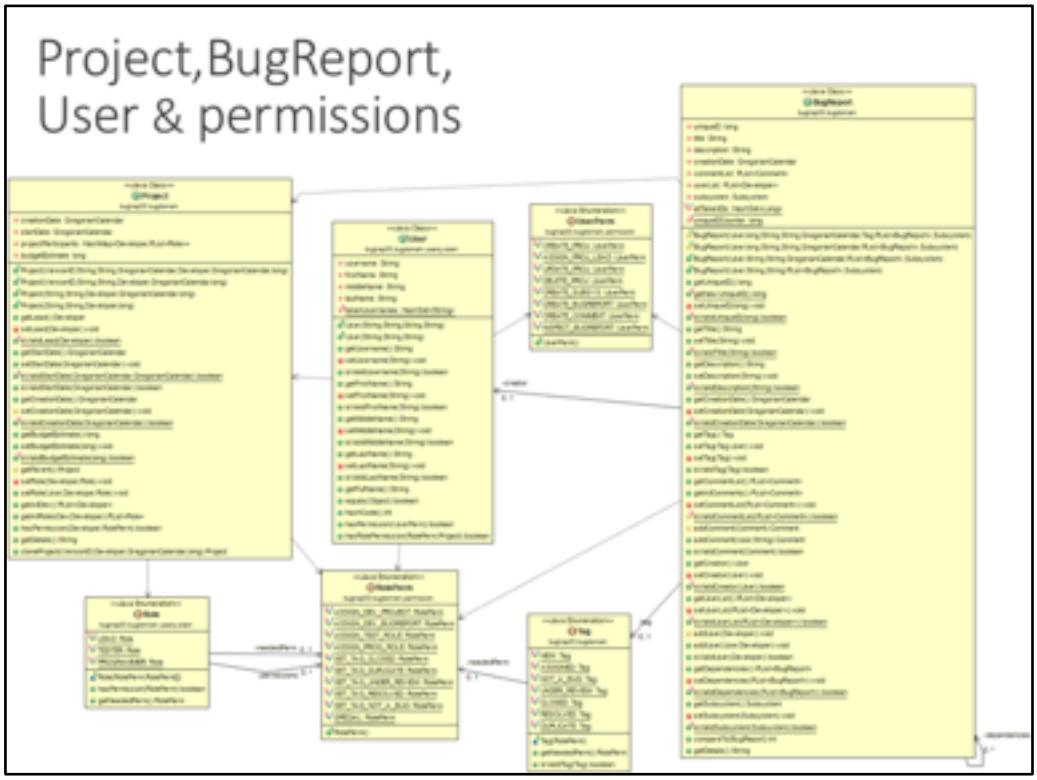
BugReport + Tags (Duplicate specifiek)

* Tag enum w logic inside the Tag enum.
+ Information Expert
+ High Cohesion

* Tags als objecten w logic inside the Tag
+ Polymorhispm -> seperate the logic inside the Tags
- High coupling between Tag Objects
+ Option to add non-static variables.



B



B
Shows Permissions interaction
Also mention the User, Dev, Admin, Issuer hierarchie.

User + Project + Role + RolePerm
* Project holding list of Users with roles. And associate list of rolePerms with Role
- High Coupling (to RolePerms; no interface of methods such as
+ Information Expert

* RolePerms = classes held in user
- Low cohesion
+ Information Expert

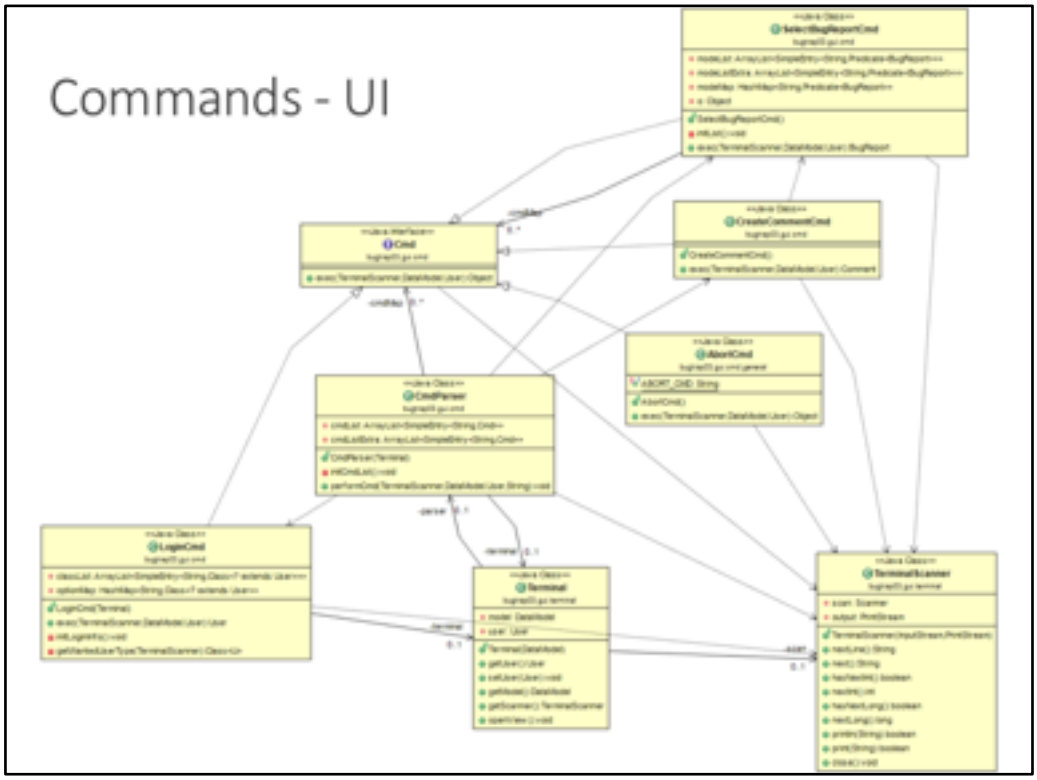
* PermissionManager - see User hierarchie + Userperm

User hierarchie + UserPerm
* UserPerm as enum and User hierarchie has arrays of the permissions the user has
+ Polymorpishm (also very flexibile can be changed dynamic)
+ High Cohesion
+ Information Expert

- High Coupling between 2 classes (A lot of coupling w permissions since User does not define methods for each permission, rather there are permissions for every action and it is checked if the user has this.)
- High Coupling between User & Permissions (Adding a permission requires an update for the user hierarchie to make sure certain users have the permissions and others do not.)

* User class + Enum userType + Using UserPerm as argument of userType
- Low Cohesion (Hard to enforce User type as parameter etc + instanceof hierarchie not possible)
- High coupling
- No Polymorpishm
+ Information Expert

* PermissionManager holding Permissions for each user with Permissions as Objects.
- High Coupling (need reference to this to check permissions)
+ High Cohesion
+ Pure Fabrication ('outsource' responsibility)
+ Polymorpishm (allow this; but isn't used)
- Polymorpishm (Lot of overhead; Big hierarchie of permissions)

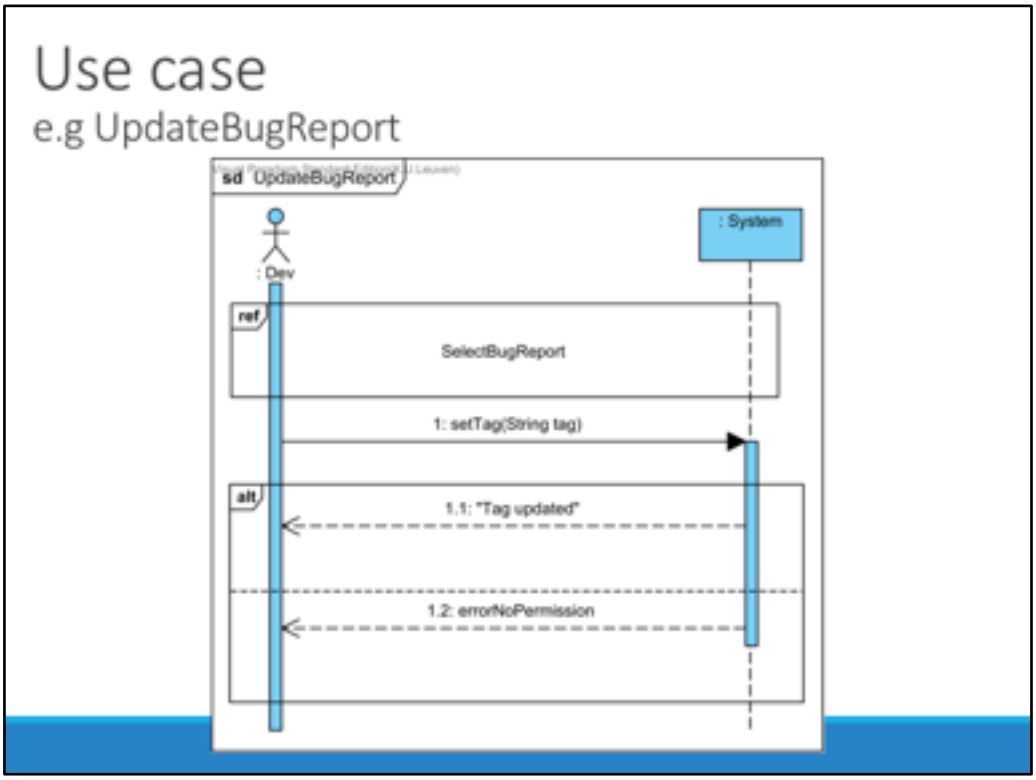


V
Explain Scenario's; Interaction from UI and How DataModel fits in.

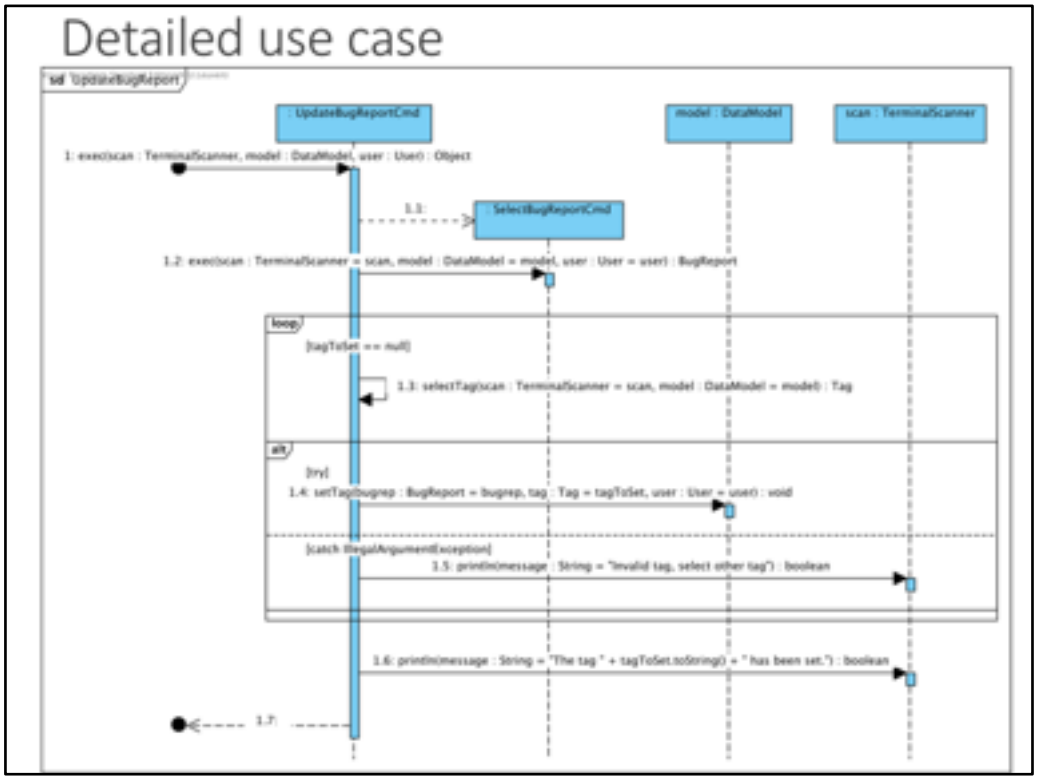
UI - Model - Cmd

* Seperate Use case steps into Cmds
+ Low coupling
- High cohesion
+ Controller
+ Polymorphispm (little bit - Cmds)

* Methods lots of methods
+ High coupling
- Low cohesion

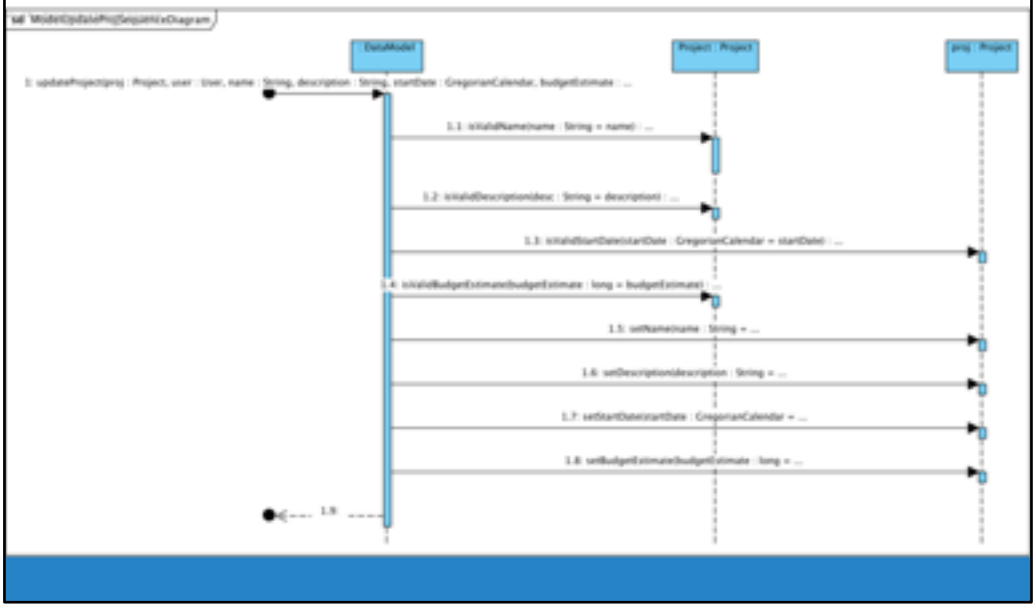


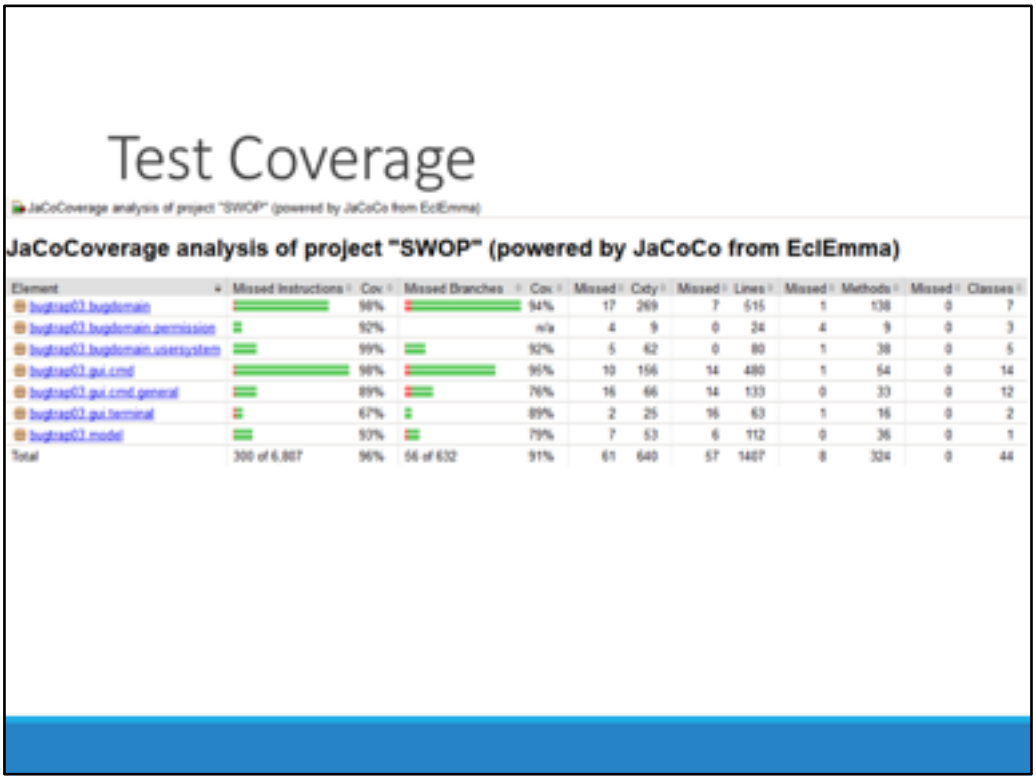
V
System sequence diagram



V
Sequence diagram

DataModel UpdateProjectSequence





K /
TODO? Tekstvak met uitleg Test strategy? (White box testing + mocking)

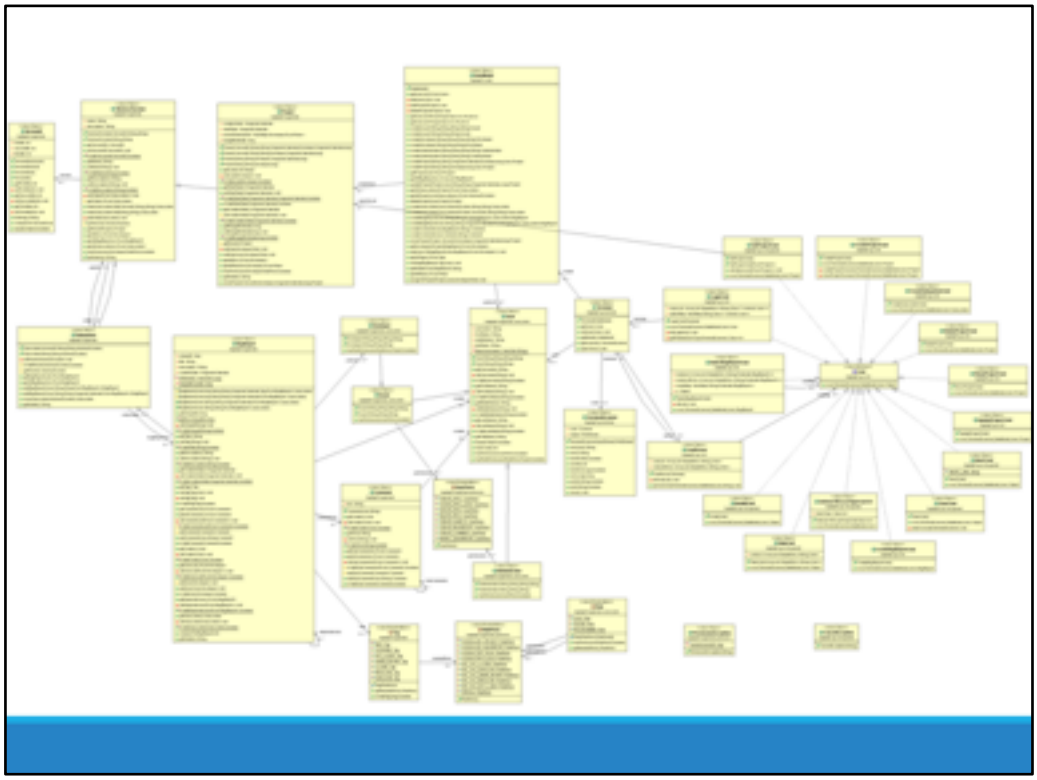
TimeTable

~~TODO Add Uurbestuding hier!!~~

Enkel totaal overzicht. NIET UITLEGGEN tijdens presentatie.

Roles

- Design Coordinator - Derkinderen Vincent
- Testing Coordinator - Buytaert Kwinten
- Domain Coordinator - Dekempeneer Mathias



Hide – Backup/Just in case

Side Information GRASP:
High cohesion
Low coupling
Information expert
Don't talk to strangers
Polymorpishm
Controller
Pure Fabrication (Not in conceptual model, specifically made for code)
Protected Variation (Assigning responsibility in a way that variations do not have undesirable effects. The way to build a stable 'interface' in a context of variations.)
Indirection (outsource responsibility)