After that we looked into the codebase.

Very quickly, we use IntelliJ Dependency Matrix to see the dependencies between packages and classes. Actually, there are quit a few cycles between components.

For example, this red box means there are 262 dependencies from analytical to gui, and this one says there are 282 backwards from gui to analytical. This is a quite strong cycle.

So we expanded the matrix at this package, to analyze it at class level. We found out that ExactModel class is the responsible for this cycle.

These dependencies caused coupling effect which makes it very hard to reuse these two packages separately.

Beside this one, there is another strong cycle between gui and engine which also has something to do with ExactModel class. We analyzed and fixed this problem and the solution will be given by Juno later.

So here is the dependencies or structure of critical components showing how different responsibility components are linked together. Arrow means there is a method call or reference, the red one means there is a cycle.

In gui package: The exact wizard is the main ui, the panels package has reusable ui components. And exactModel is the algorithm model. Solverclient invoke model solver and start threads.

Jobs from solverClient will be passed to SolverDispatcher. and then they are initiated and delegated to some particular solver class.

Engine package contains simulation engine classes for JMT apps.

Common package has exception classes and setting classes.

Framework package has xml utilities and base classes for gui components.

These red arrows going from and to ExcatModel indicates this class is involved in two cycles.

Next, Juno will show you how to improve the structure.