BOOK IA - INSULIN PUMP (MODULE A)

Use Case Packages with the Executable MBSE Profile

16 MAY 2024
OPEN LAB TUTORIAL
RHAPSODY V9.0.1
EXECUTABLE MBSE PROFILE V4.3.D.RELEASE

www.mbsetraining.com

Trademarks and the rest

- IBM, Rhapsody, DOORS, Rational are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.
- Microsoft, Windows, PowerPoint, Excel, MS-DOS are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- SysML, OMG Systems Modeling Language, OMG, Object Management Group, Unified Modeling Language either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.
- Other company, product, or service names may be trademarks or service marks of others.
- Images used under license from 123RF.com: gunnar3000©123RF.com, serknor@123RF.com, cocoo@123RF.com, lenanet@123RF.com

Table of Contents

Introduction

- This lab book is one of two workshop lab books that build the insulin pump system project from scratch using the open-source Executable MBSE profile written by MBSE Training and Consulting Ltd
- It uses version 4.3.d.Release of the SysMLHelper: http://www.executablembse.com/
- The SysMLHelper is licensed under GPL 3.0 which is a copyleft license, meaning that any copy or modification of the original code must also be released under the GPL v3
- The SysMLHelper is entirely self-contained Java, a conscious decision was made not to use third-party or open-source libraries/.jars other than a standard Java runtime (incl. Swing for GUI) and the Rhapsody Java API .jar
- It may take to 1-2 hours to complete this lab book, assuming everything is correctly installed

CREATE AN EXECUTABLE MBSE PROJECT

LAB IA1

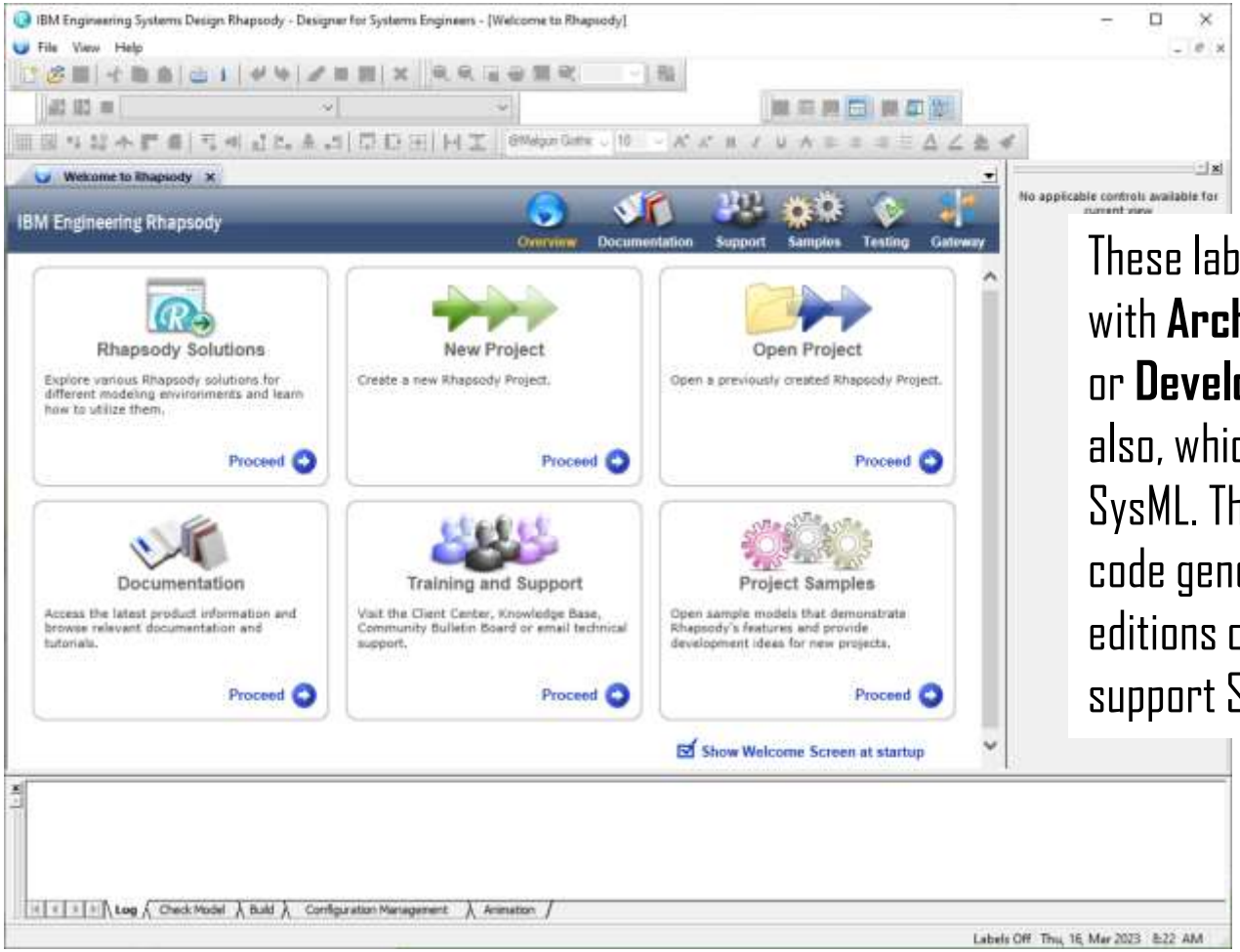
"INSULIN PUMP" CASE STUDY

LAB IA1

"hear and I forget. I see and I remember. I do and I understand" (Confucius 551 BC - 479 BC)

Launch Rhapsody

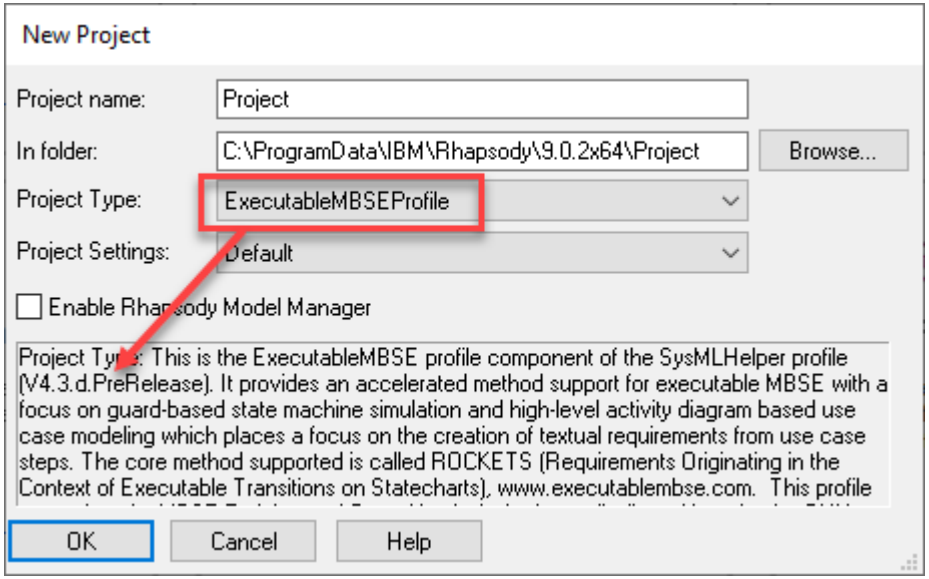
- Launch Rhapsody's **Designer for Systems** or **Architect for Systems** Edition



These labs could be done with **Architect for Systems** or **Developer** for C/C++ also, which also support SysML. The labs don't use code generation and all the editions of Rhapsody support SysML profile usage

Check version of ExecutableMBSE Profile

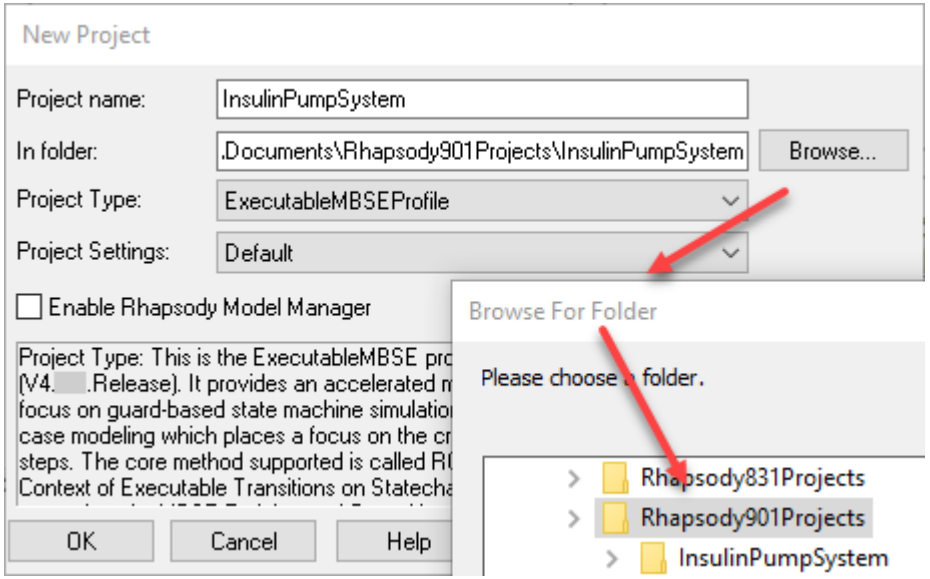
- Launch the new project dialog
- Select **ExecutableMBSEProfile** as the **Project Type** and name the project **InsulinPumpSystem**. Profiles are a modeling extension mechanism in UML



Check that version number says **V4.3.d.Release** as this will match the functionality used in this guide. This guide works with earlier versions of Rhapsody than 9.0.1 but some screenshots may differ (e.g., display properties of parts when dropped on diagrams)

Browse to an appropriate folder

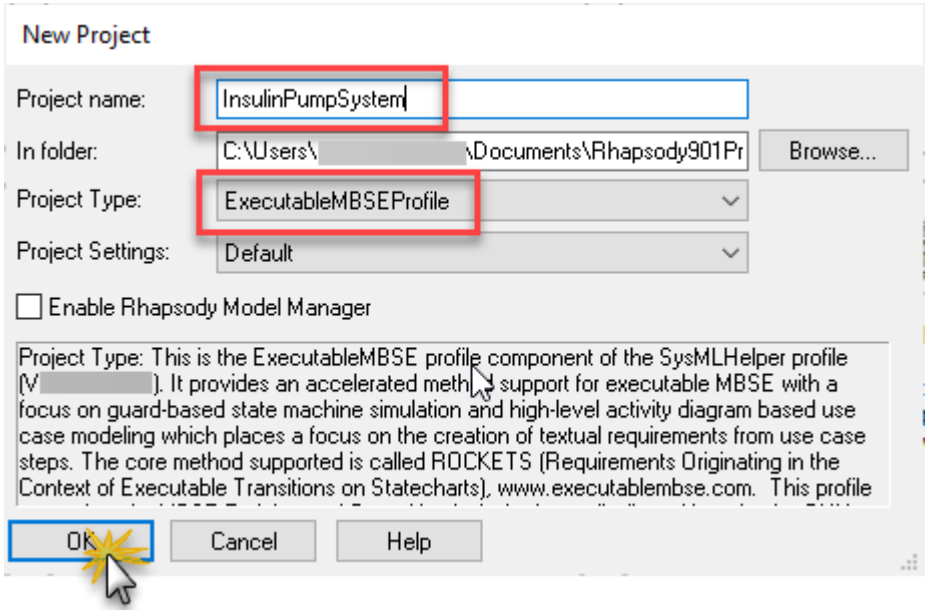
- If necessary, click **Browse…** for the **In folder** and ensure model will be saved in an appropriate read/write location for the user, and not part of the installation



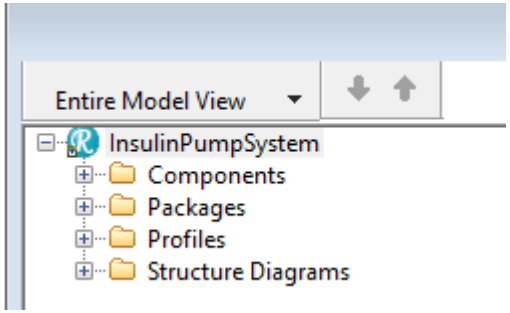
Adding ProjectsDirectory=<Path> to [GENERAL] section of Rhapsody.ini can avoid having to do this (ensure Rhapsody closed). This screenshot will vary with yours and shows a Rhapsody901Projects folder in My Documents. Choose Desktop, if not sure where to put project. Rhapsody will create folder underneath this

Create an InsulinPump project

- Create a project called **InsulinPumpSystem** with the **ExecutableMBSEProfile** and **Default** settings
- Click **OK** to create the project



Don't enable Rhapsody Model Manager. It is always best to do this at a later point anyway, particularly if you want different unit (model file) granularity

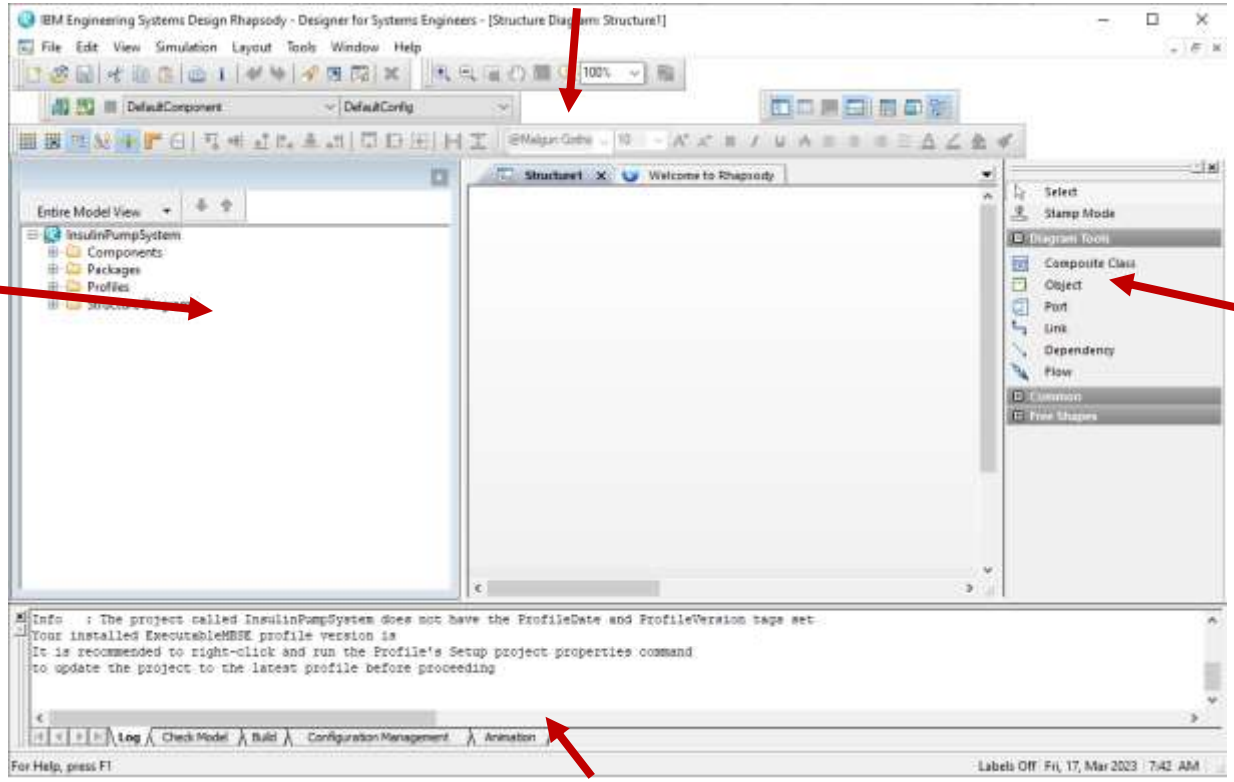


The Rhapsody workspace

- You can now see the main Rhapsody workspace. By default, the browser is docked left, diagrams are docked centrally, and the drawing toolbar is docked to the right

Diagram tabs

Browser



Drawing toolbar

Output Window panes

View the Log window

- View the **Log** window

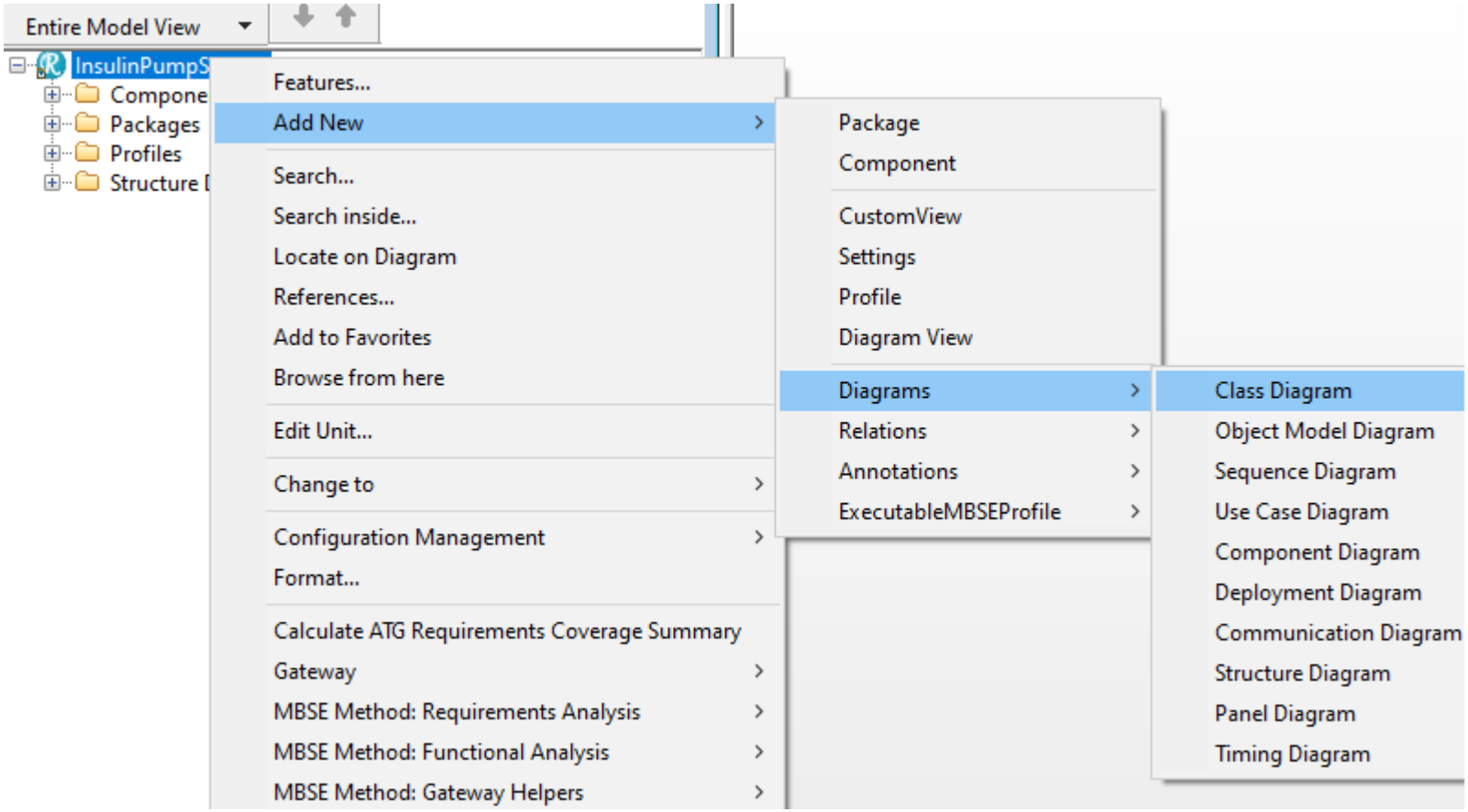
```
SysMLHelperPlugin is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even
New right-click 'MBSE Method' commands have been added.
Info   : ExecutableMBSE_RPApplicationListener was loaded - Listening for events (double-click etc)
Info   : The ExecutableMBSE profile version is
Info   : The project called InsulinPumpSystem does not have the ProfileDate and ProfileVersion tags set
Your installed ExecutableMBSE profile version is
It is recommended to right-click and run the Profile's Setup project properties command
to update the project to the latest profile before proceeding
```

Log | Check Model | Build | Configuration Management | Animation

The ExecutableMBSE profile includes a Java plugin that adds new menus to assist with method automation. The Log window contains an indication that this plugin has loaded plus which version it is. If you don't see this, then it's probable your $OMROOT/Profiles/SysMLProfile folder is not in the correct location! This must be fixed before proceeding

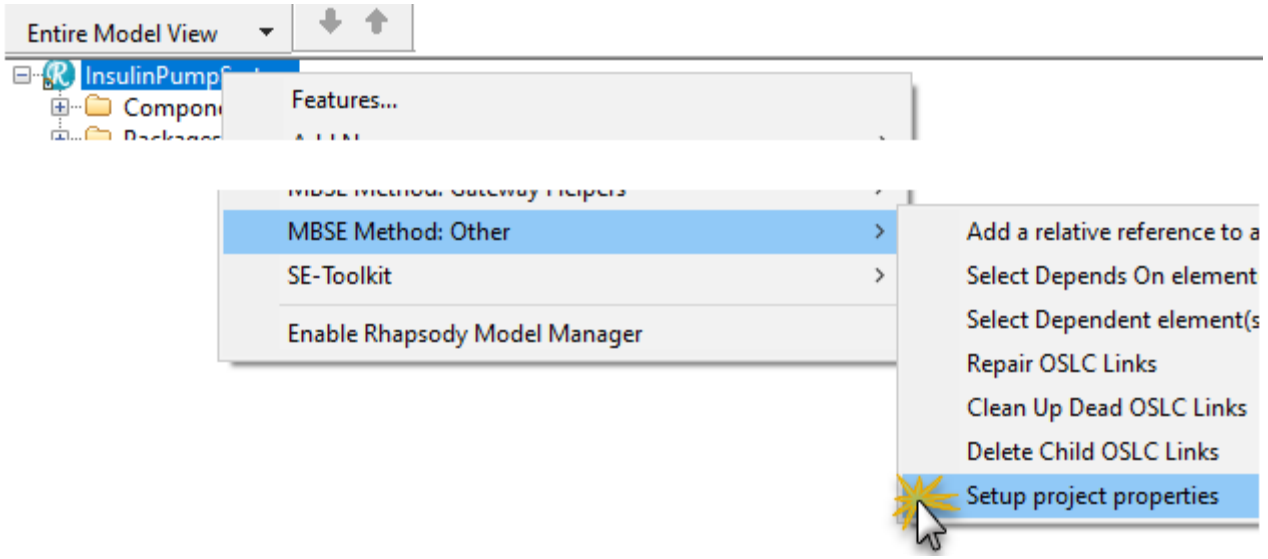
View the unmodified Add New menus

- Right-click the project and view the **Add New** menus. These are currently set to the UML diagrams and elements



Run command to Setup project properties

- Right-click the root element in the project and choose **MBSE Method: Other command > Setup project properties**



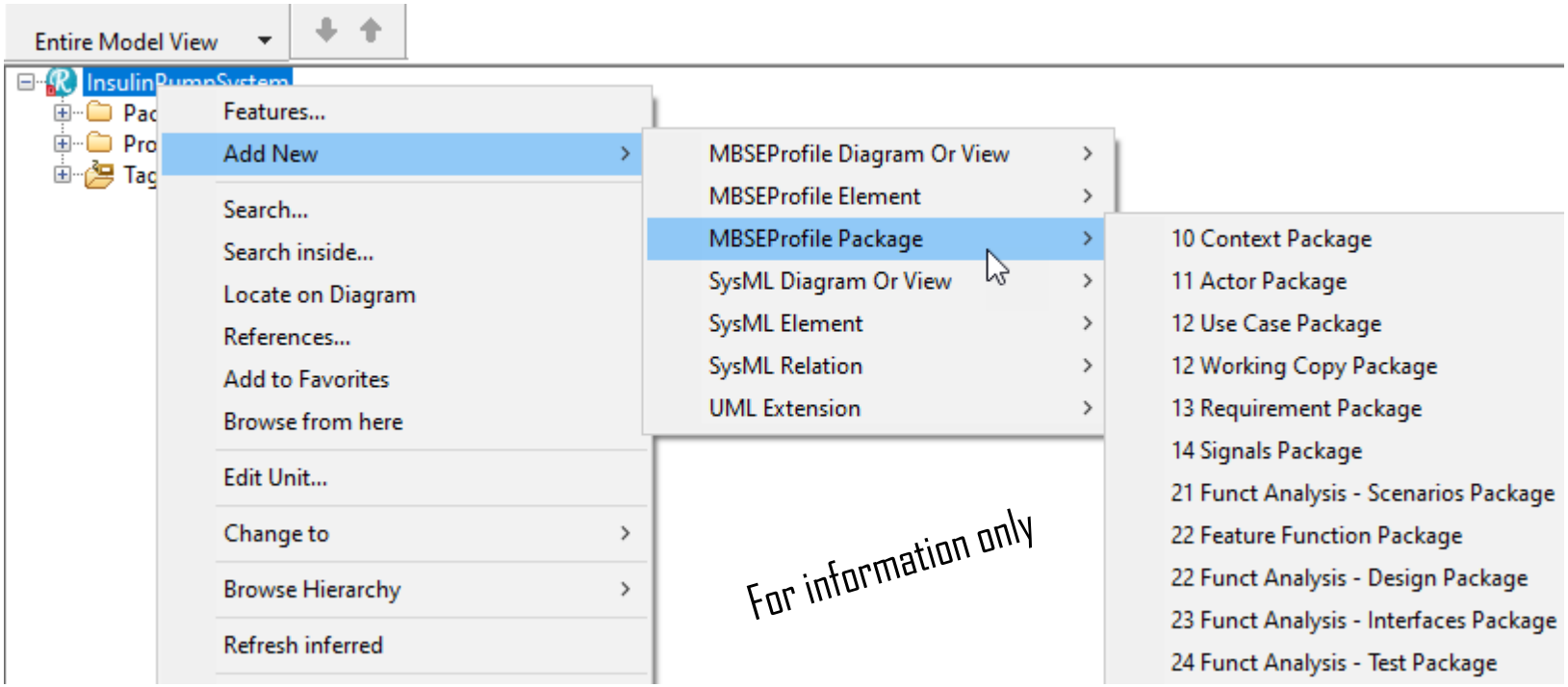
This command only needs to be run when setting-up a project or updating a project with a new version of the ExecutableMBSE profile. It will set up some direct project properties and add a date stamp tag used to advise other users on the version of profile they need

Click Yes to update the project properties

- Click **OK** to close the information dialog and then **Yes** to update the project's properties

Right-click to view Add New menus

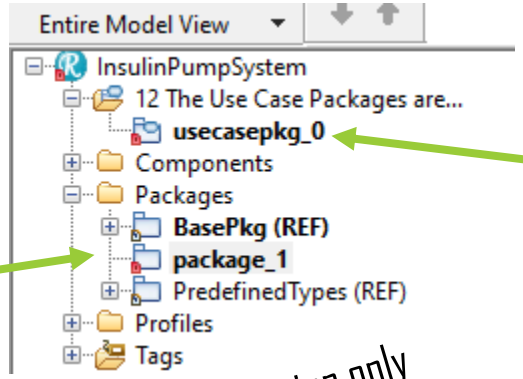
- This has changed the right-click menus to have Executable MBSE ones in addition to SysML. It does this by overriding the **General::Model::AddNewMenuStructureContent** property on the model. There are several new term package types used in the method



New Terms

- A **New Term** in Rhapsody is a stereotype that has been set-up in a way which overrides the user interface, including creating new categories (yellow folders) in the browser. Usually, it has a new icon and appears as a type in drawing toolbars and right-click menus. Using them is like doing “fancy dress”, we can dress things up to look and act differently
- Example:

UML/SysML have the base language concept of a Package. It has an unadorned blue folder icon



For information only

This package has been changed to a UseCasePackage (a new term stereotype in the Executable MBSE profile). Properties have overridden the folder’s icon and category name

Benefits of using New Term packages

- The benefits of using New Term packages include but are not limited to:
  - Unique icons convey understanding of package contents
  - Can limit menus to a smaller set of element types. This improves consistency of model structures across different models and provides a separation of concern when working as a team
  - Adding Add New menus for new element types and views such as tables and matrices that are specific to different types of package
  - Usability features can be inferred from relationships such as dependencies, for example, a dependency from a use case package to a requirements package can be used to infer the auto-move of requirements into the requirements package

CREATE A USE CASE PACKAGE

LAB IA2

"INSULIN PUMP" CASE STUDY

LAB IA2

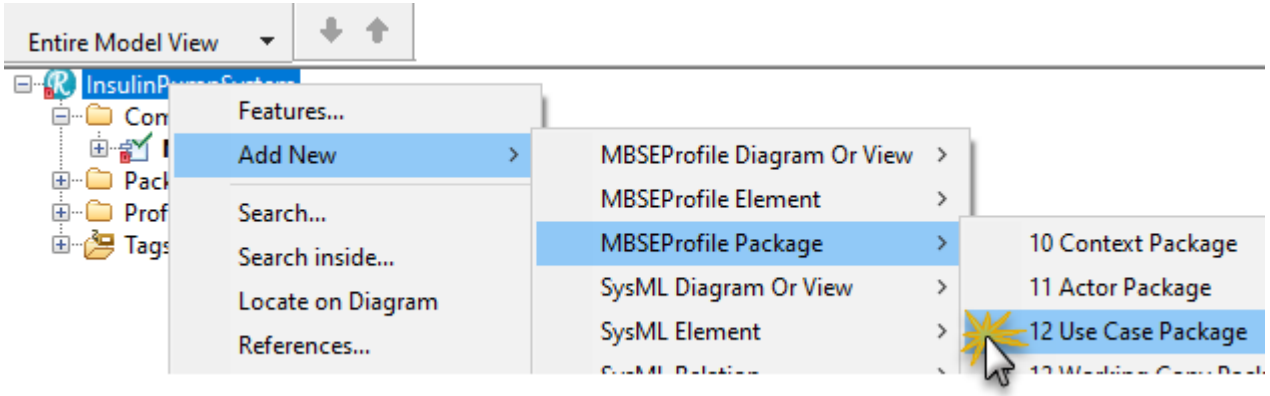
"I hear and I forget. I see and I remember. I do and I understand" (Confucius 551 BC - 479 BC)

Use Case Diagram

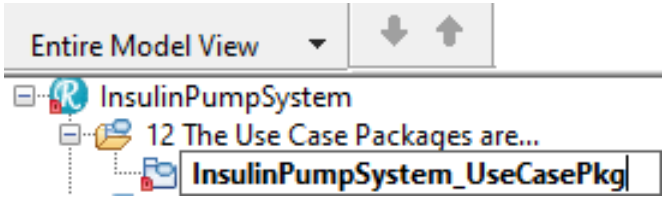
- In this lab we're going to create a simple use case diagram. Use cases are a great way to have conversations with stakeholders about how a system should work
  - **Use Case**: A set of steps including alternate flows that provides some value to an actor. Often a verbal-noun naming convention is used
  - **Actor**: The role played by an external entity that interacts with the system to gain value for itself or in the provision of that value to other actors
  - **Association**: A relationship that shows that an actor participants in a use case
- Use cases are a description of the behaviour performed by the system. The use case diagram does not provide this description, as we could choose to use many ways to describe the behavior from textual statements, to activity diagrams, to sequence diagrams
- They can bridge the gap between what the user needs and how the system provides what the user needs

Add a use case package

- Use the **Add New** right-click menu on the project to create a use case package



- Call it **InsulinPumpSystem_UseCasePkg**



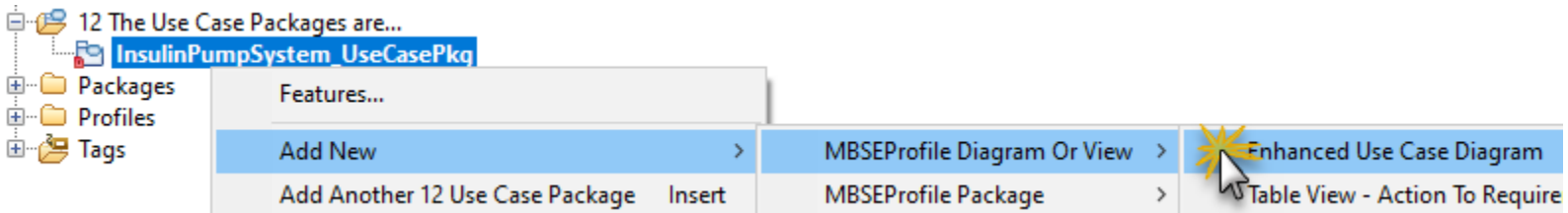
Using package name conventions

- We will use a post-fix naming convention <u>without spaces for packages</u>:

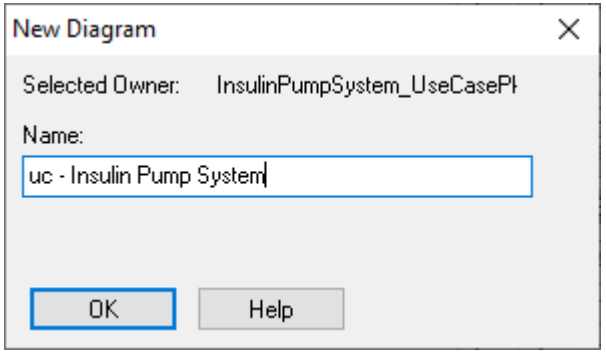
New Term type	Post-fix	Description
Use Case Package	_UseCasePkg	A container for use case diagrams and use cases being developed by a particular requirements analyst.
Context Package	_ContextPkg	A container for context diagrams and associated element usages and flows.
Actor Package	_ActorPkg	A container for actors shared by other packages.
Requirement Package	_RequirementPkg	A container for requirements and associated table or matrix views.
Signal Package	_SignalPkg	A container for shared signals (captured as events) used on context diagrams or used as flows by function or interface blocks.
Feature Function Package	_FunctionPkg or _FeaturePkg	A container for developing the sub functions of either a (system-level) feature or a function (below system level).
System Architecture Package	_ArchitecturePkg _LogicalArchitecturePkg or _PhysicalArchitecturePkg	A container for developing the structural system hierarchy.

Add an Enhanced Use Case Diagram

- Right-click the use case package, and choose **Add New > MBSEProfile Diagram Or View > Enhanced Use Case Diagram**



- Call the diagram **uc – Insulin Pump System** (we will use spaces in the name of diagrams)



We will name all diagrams/views with a prefix. The profile will automatically pre-fix with **uc –** (uc is the SysML abbreviation for a use case diagram). This convention significantly assists usability when trying to spot different diagrams in the diagram tabs and menus

Using diagram name conventions

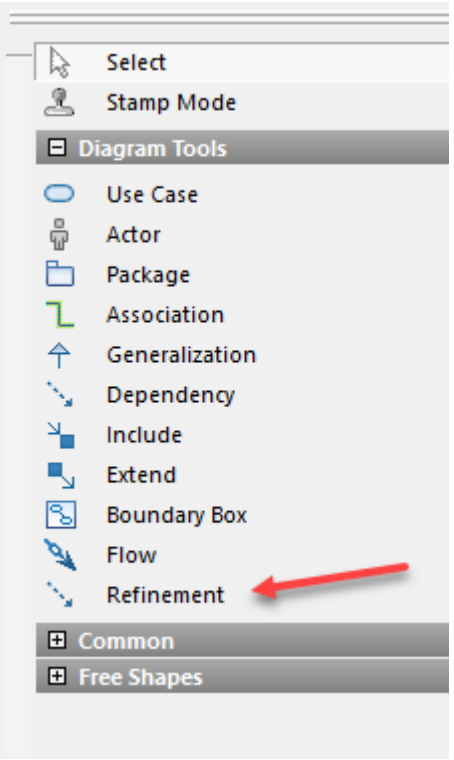
- The following is the lower-case pre-fix naming convention for diagrams:

Diagram type	Pre-fix	Description
Use Case Diagrams	uc -	A diagram of use cases and their associated actor relations
Context Diagrams	ctx -	A profile specialised form of ibd for showing system context with flows to actors
Activity Diagrams	act -	An activity diagram. The profile adds a specialization of activity diagram called a Textual Activity designed for steps of a use case
Internal Block Diagram	ibd -	An internal block diagram. The profile adds a specialisation called internal block diagram – system with different port types and connectors for doing functional flow diagrams
Block Definition Diagram	bdd -	A block definition diagram is a SysML diagram for defining a hierarchy of blocks. The profile adds a specialization of this with different types of blocks, e.g., to differentiate systems from subsystems, and then concept of function blocks

All these diagram abbreviations, apart from **ctx** align with SysML. SysML doesn't have an explicit context diagram, rather uses the internal block diagram. The Executable MBSE profile adds customization to ease drawing simple context diagrams

Enhanced Use Case Diagram drawing toolbar

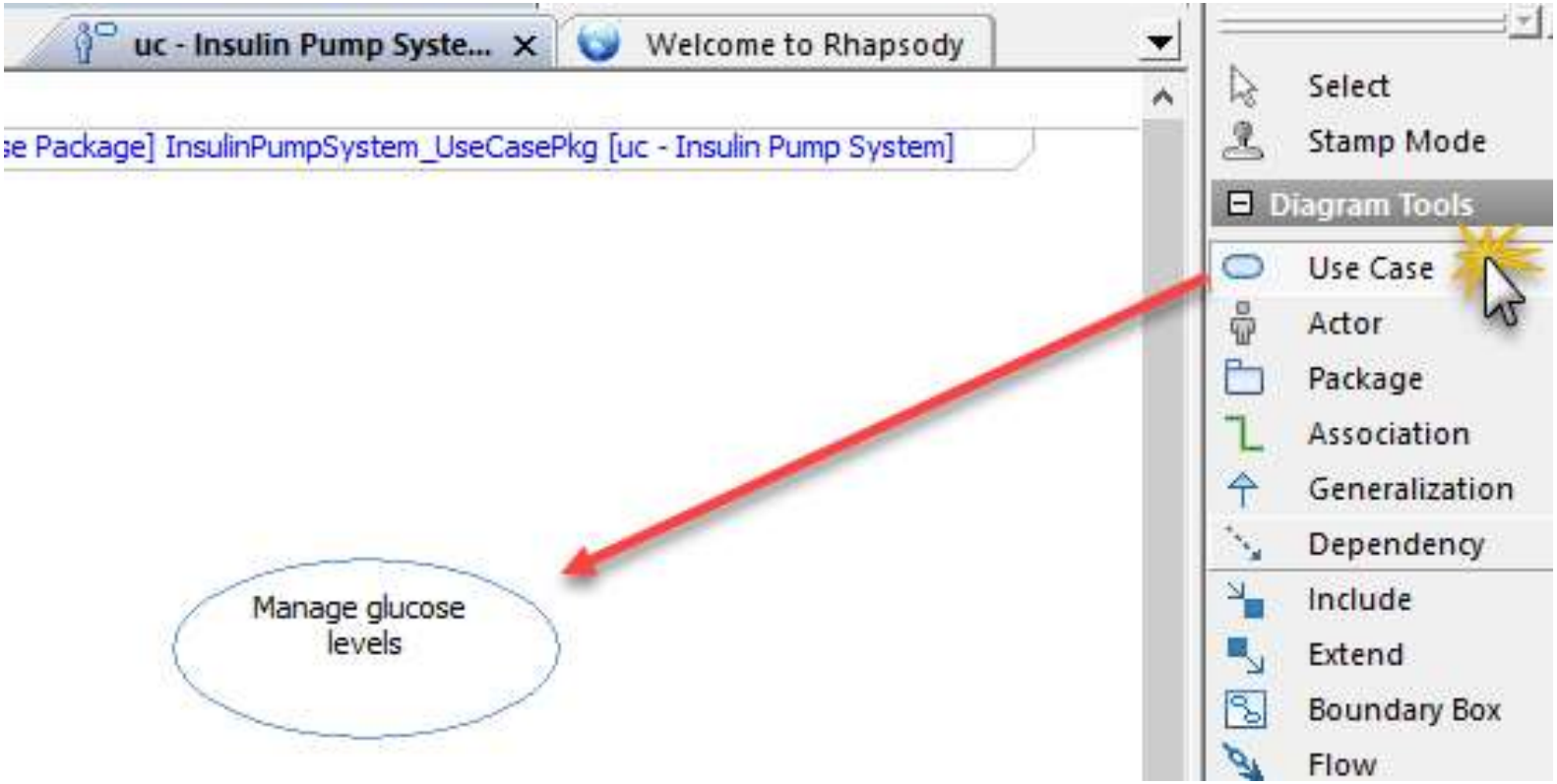
- The profile extends the factory use case diagram with an additional drawing tool to draw **Refinement** relations (dependencies with **«refine»** stereotype) directly. Other than that, it's the standard diagram when it comes to use case modeling



Refinement is the new term dependency we will use to trace use cases to requirements. Having it available in the use case diagram drawing toolbar speeds up modeling and encourages consistency and alignment to modeling choices across projects

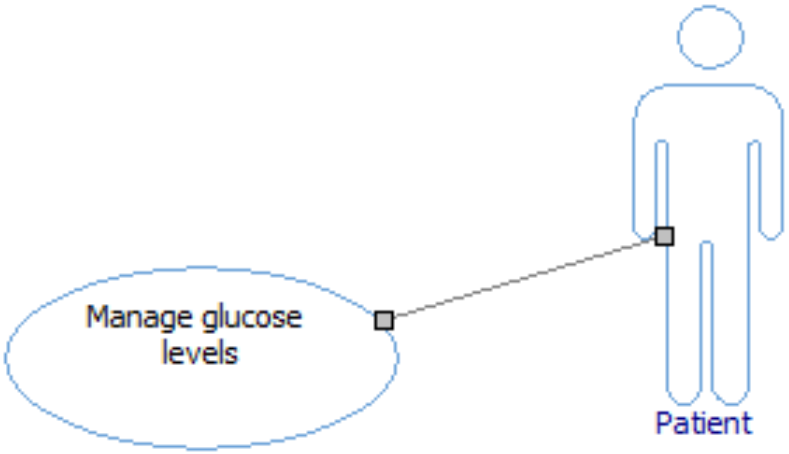
Add a use case

- Add a use case: **Manage glucose levels**



Add an actor and an association

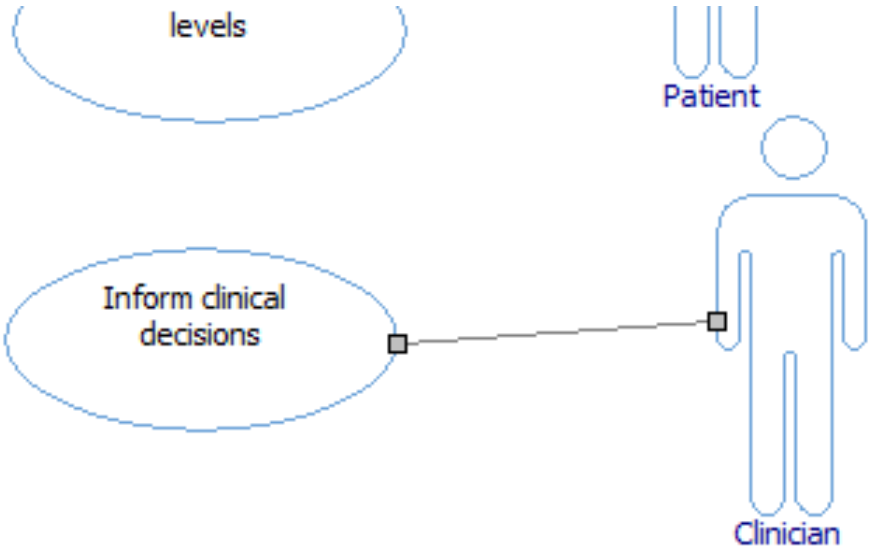
- Add a **Patient** actor
- Add an **association** between the **Patient** and the **Manage blood glucose** use case



The association tells us that the actor participates in the use case. In this case it's the primary actor for the use case as they gain the value

Add a different use case and different actor

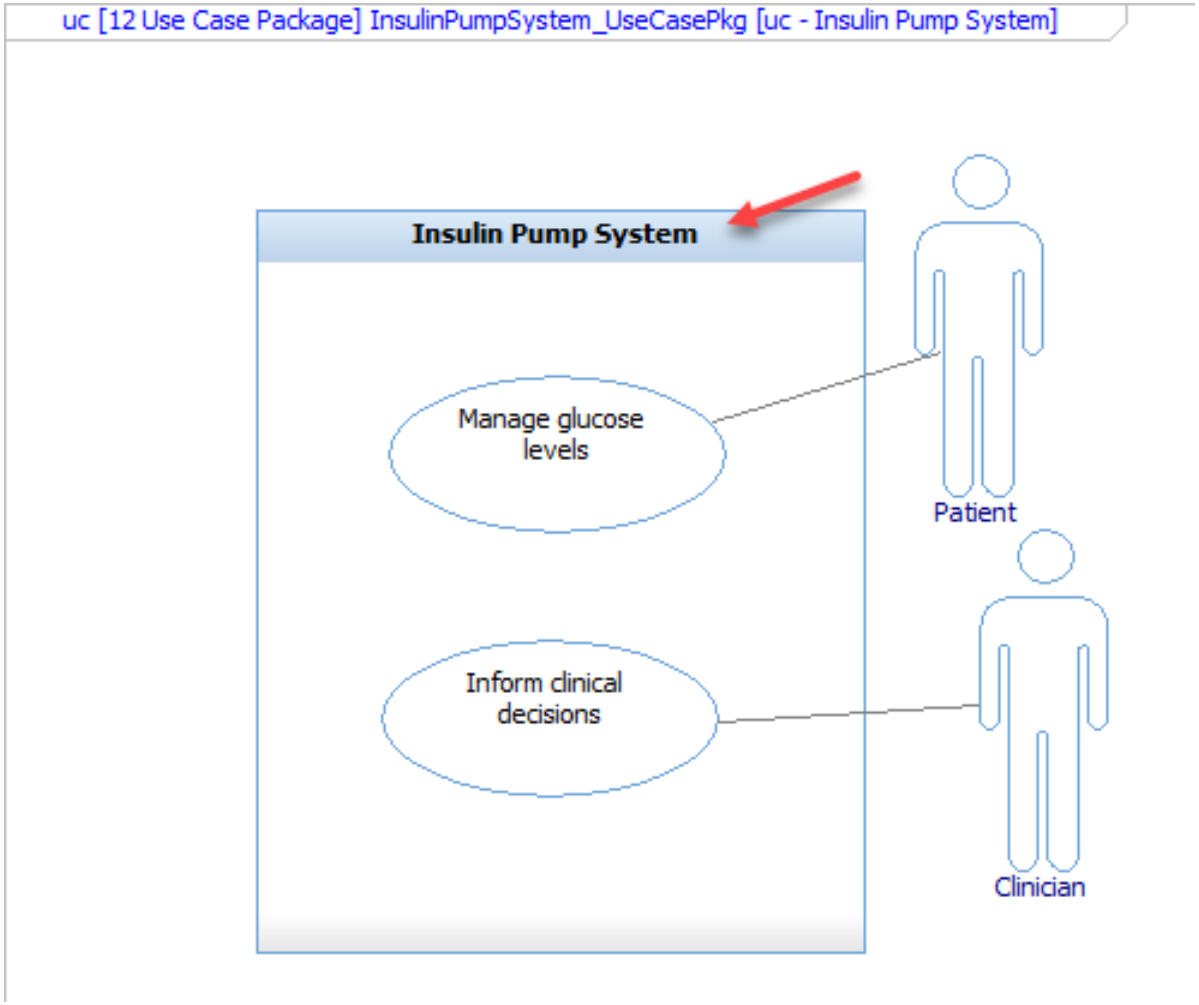
- Add another use case **Inform clinical decision** with a different actor called a **Clinician**



Actors are roles played with respect to the system by external entities where the needs are distinct, i.e. the Clinician wants very different things from the system than a Patient

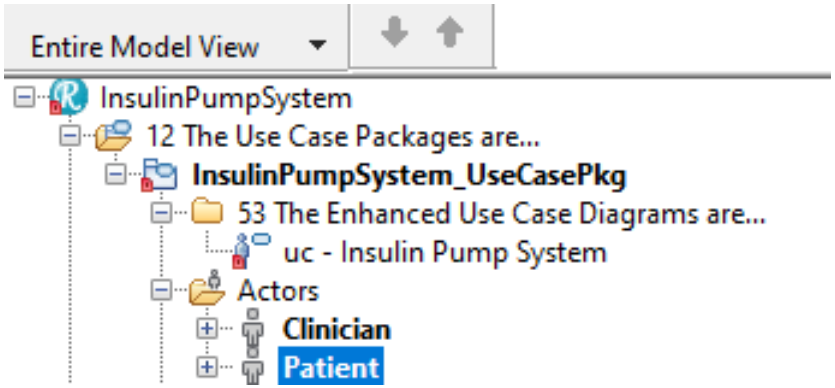
Add boundary box

- Complete the diagram with a boundary box named **Insulin Pump System**



Review of actor in browser

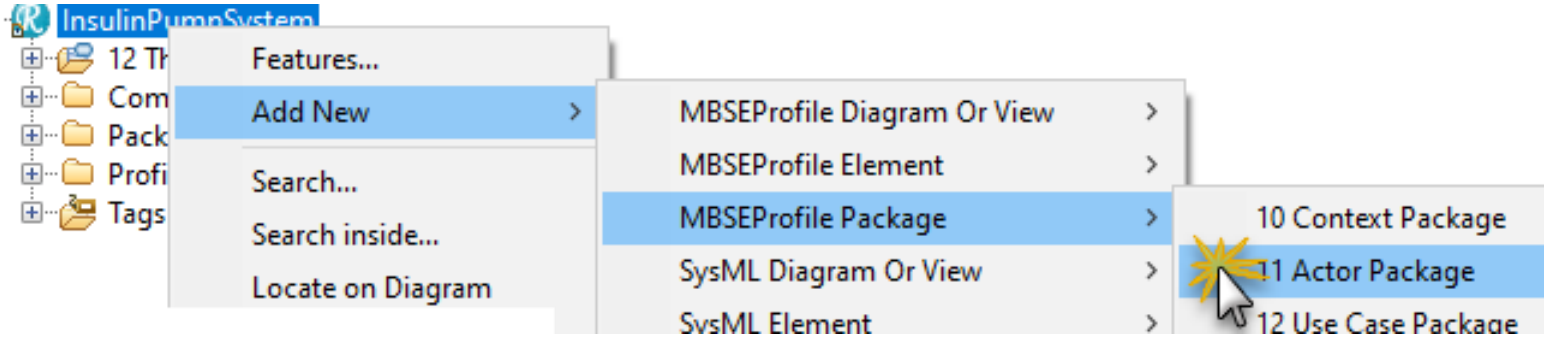
- Expand the browser. The actors we've added to the diagram are under the use case package



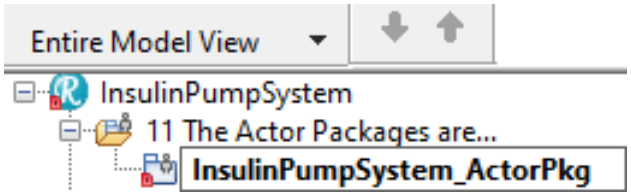
- If we have multiple use cases being worked on by different modelers then we would have use case packages for each of them, all sharing the same actors, hence we will now move the actors into a shared actor package

Add an actor package to the project

- Right-click the project root and add a new **Actor Package**



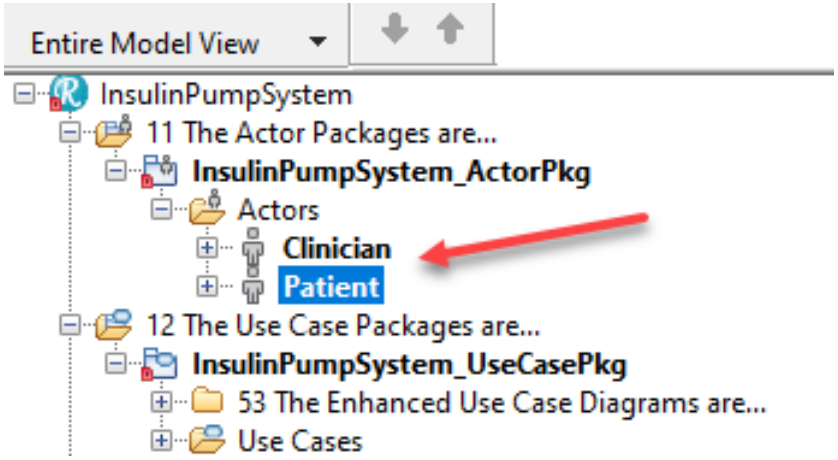
- Call it: **InsulinPumpSystem_ActorPkg**



Packages are also files on the file system (in the _rpy folder for the project). This naming convention keeps them unique and easily identifiable should we see them in a pending changes view under model management, or want to add them to other projects

Move the actors into the actor package

- Move the actors into the actor package



- **Save** the project

The unit overlay icon changes from red to black when the unit is saved

Using package name conventions

- We have used a post-fix naming convention without spaces for packages:

New Term type	Post-fix	Description
Use Case Package	_UseCasePkg	A container for use case diagrams and use cases being developed by a particular requirements analyst.
Context Package	_ContextPkg	A container for context diagrams and associated global element usages and flows.
Actor Package	_ActorPkg	A container for actors shared by other packages.
Requirement Package	_RequirementPkg	A container for requirements and associated table or matrix views.
Signals Package	_SignalsPkg	A container for shared signals (these are captured as events) used on context diagrams or used as flows by Function blocks or Interface Blocks.
Feature Function Package	_FunctionPkg or _FeaturePkg	A container for developing the sub functions of either a (system-level) feature or a function (below system level).
System Architecture Package	_ArchitecturePkg _LogicalArchitecturePkg or _PhysicalArchitecturePkg	A container for developing the structural system hierarchy.

CREATE A REQUIREMENT PACKAGE

LAB IA3

"INSULIN PUMP" CASE STUDY

LAB IA3

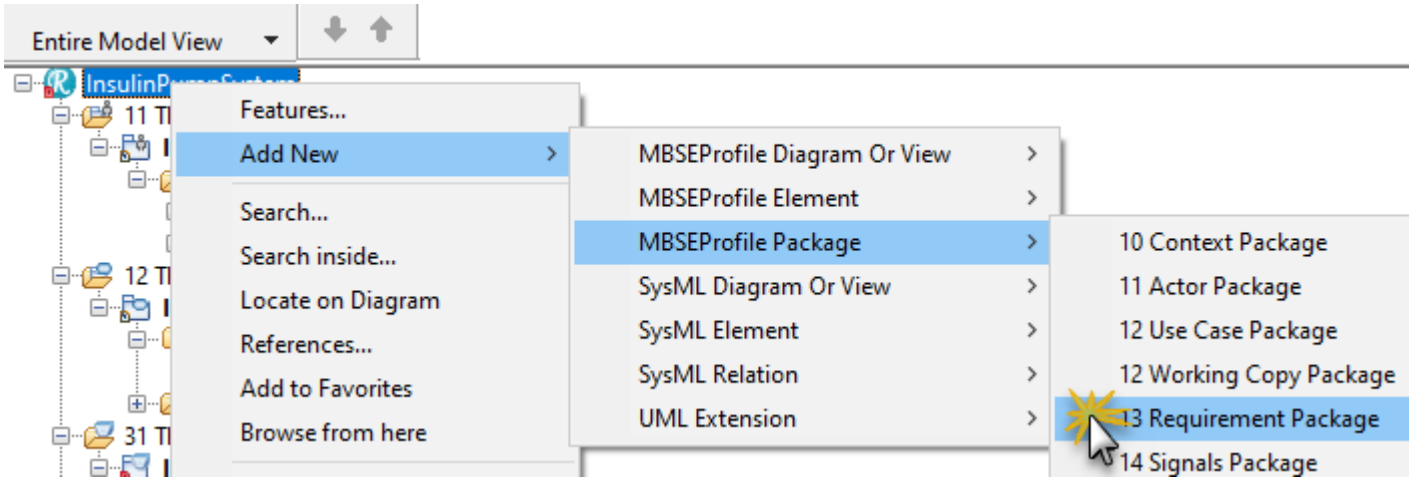
"hear and I forget. I see and I remember. I do and I understand" (Confucius 551 BC - 479 BC)

Requirements and models

- Requirements (textual statements of need) can be interweaved with model information, shown on any diagram in Rhapsody, and in tables and matrices
- We will have requirements that relate to different use cases; hence it helps to have a common package to containing them
- Normally when you add a requirement to a diagram, the requirement will be placed in the browser under the diagram element or the package that owns the diagram.
- The profile has an automation helper that will move requirements into a requirement package for you. This will be enabled by default and works based on dependency relations that a modeler adds to the project

Add a Requirement Package to the project

- Add a **Requirement Package** to the project



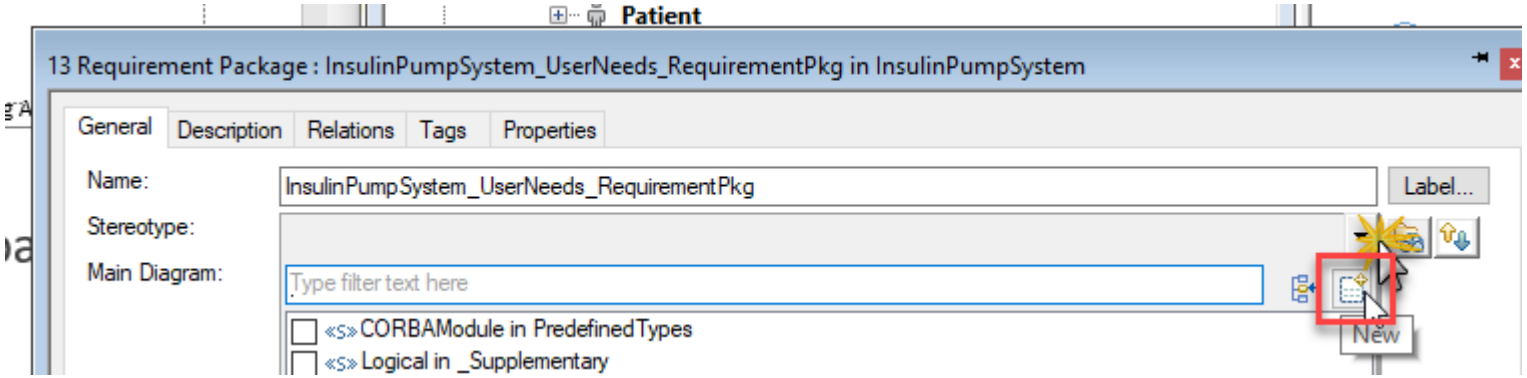
- Call it **InsulinPumpSystem_UserNeeds_RequirementPkg**

We will have different requirement packages for different requirement layers or groups; hence we will make its contents obvious in the name (hence the _UserNeeds_ bit)

Create a new stereotype on package

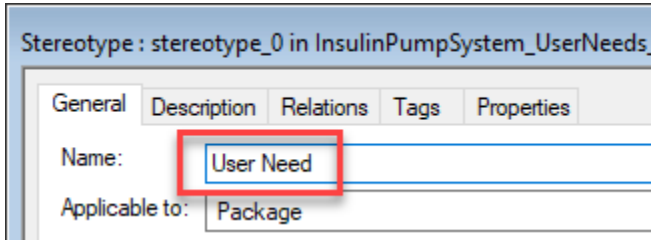
- We can get the Executable MBSE helper to stereotype requirements automatically. Open the **Features…** for the requirement package

- Select the pull-down list for **Stereotype** and press the **New** button to create a new stereotype



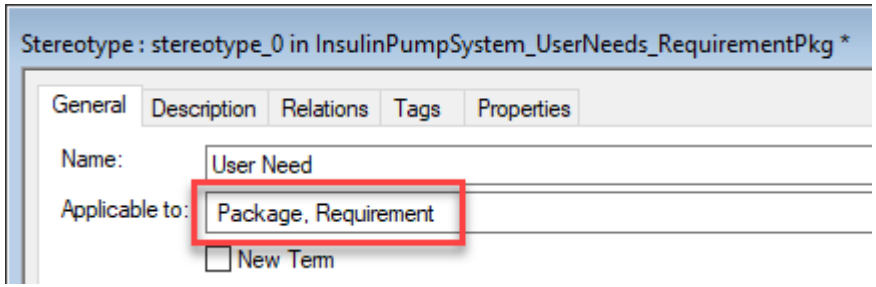
Rename stereotype to User Need and set applicable to

- Call the stereotype: **User Need**

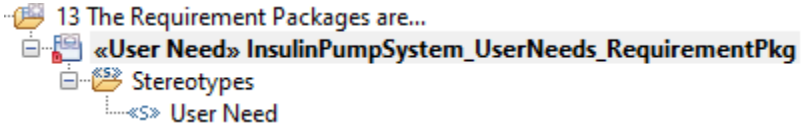


When adding new requirement stereotypes do not make them New Terms as they can't be applied to more than one element type, and some things like Check Model may not work

- Make it applicable to **Package** <u>and</u> **Requirement** meta-classes

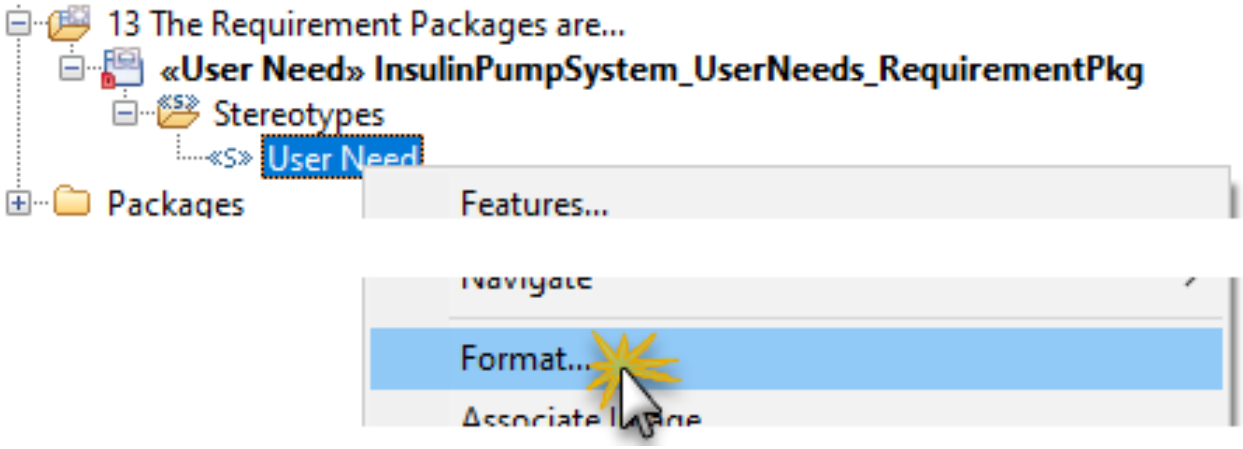


- You can now close the Features dialog for the stereotype <u>and</u> the package



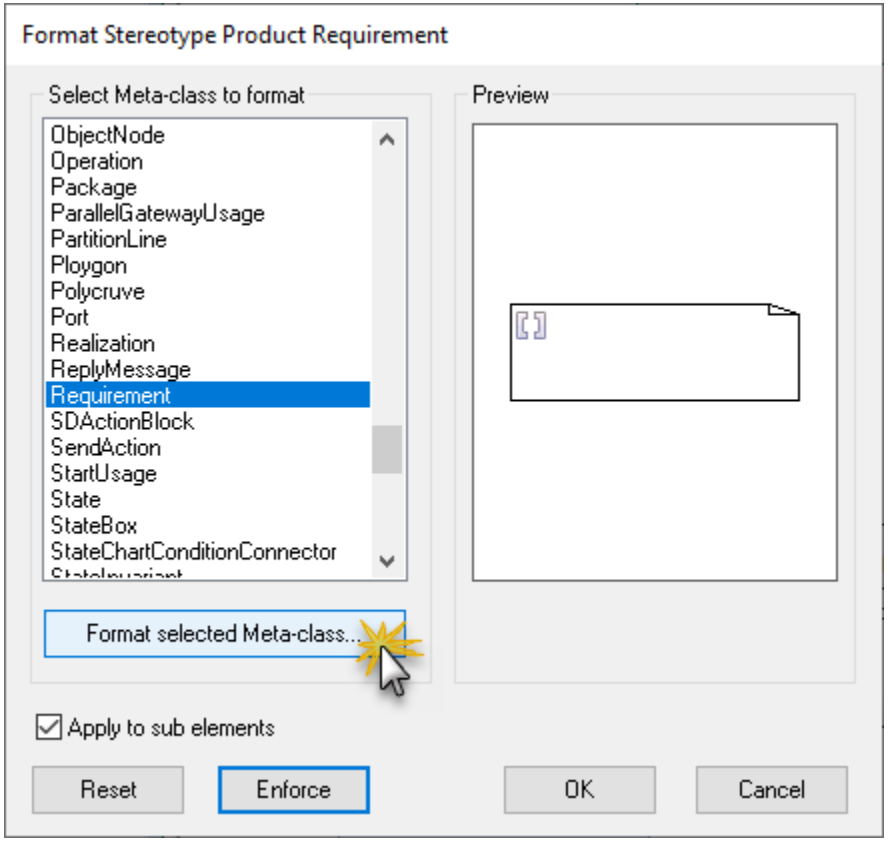
Choose to Format the stereotype

- We can associate a format to requirements whenever the stereotype is applied
- Locate the **User Need** stereotype, right-click and choose **Format…**



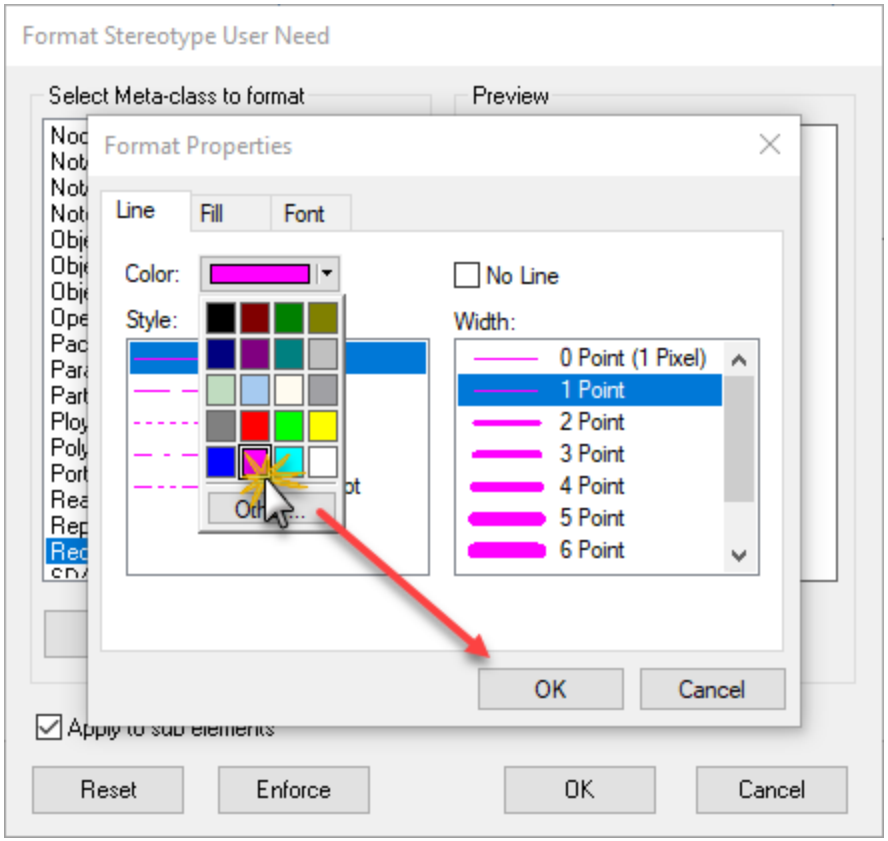
Format the Requirement meta-class

- Select **Requirement** meta-class and choose **Format selected Meta-class…**

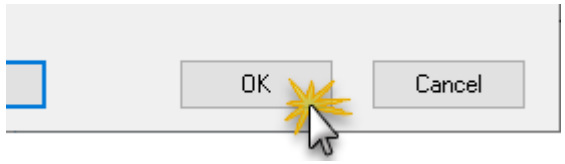


Choose line color of green

- Choose line color of pink and click **OK**

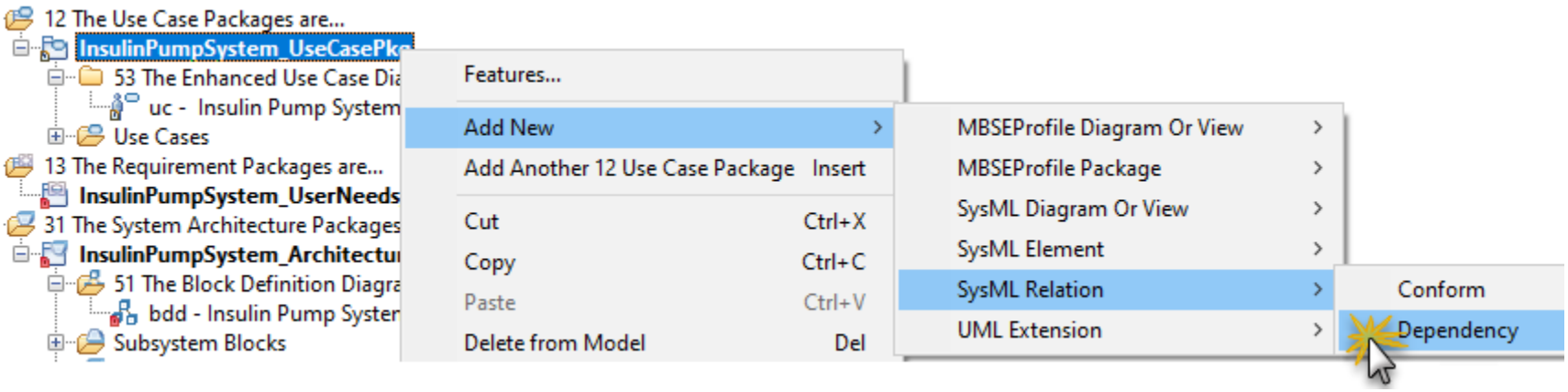


- Click **OK** to close the Format Stereotype dialog



Add a Dependency to the requirement package

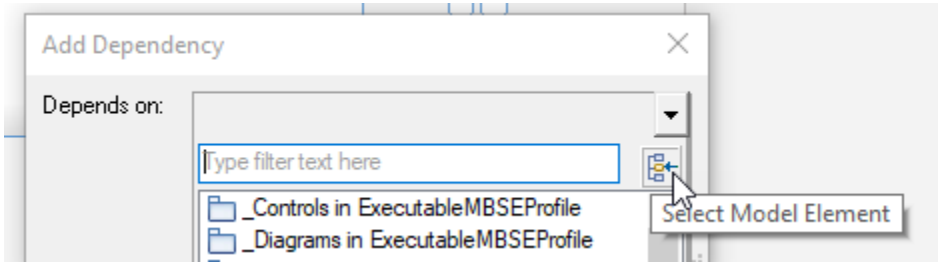
- Locate the **_UseCasePkg**
- Right click it and choose **Add New > SysML Relation > Dependency**



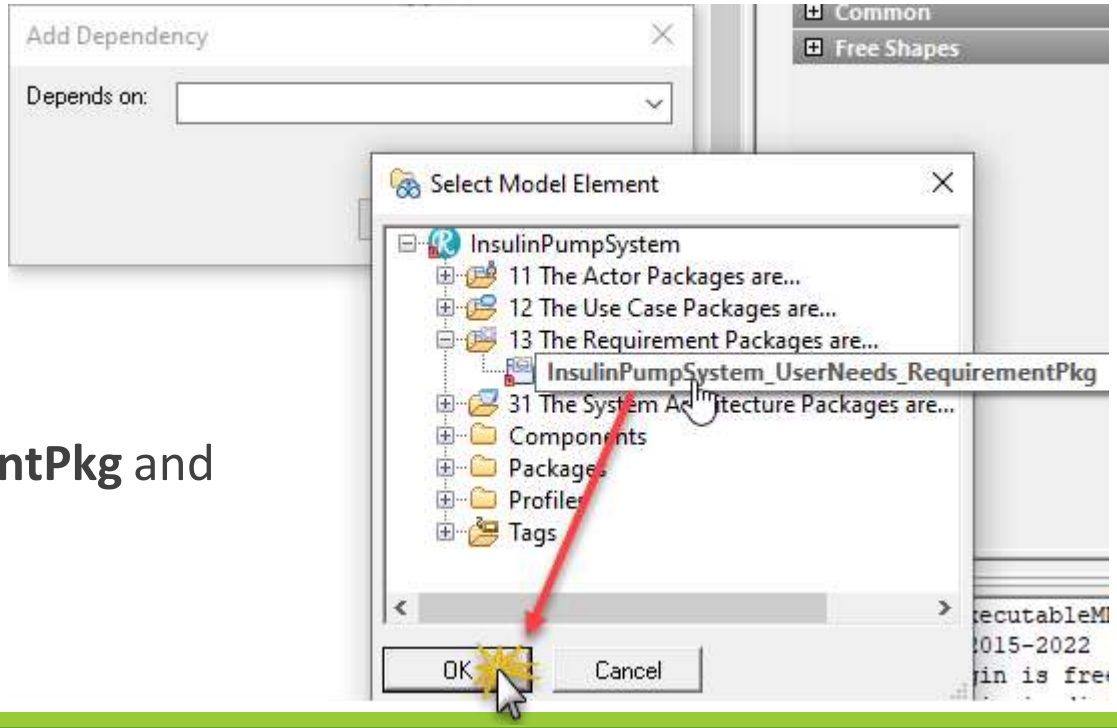
We will discover that these steps can all be automated, but it's worth knowing what they are to explain how it all works

Add a Dependency to the requirement package

- Click the icon to **Select Model Element**

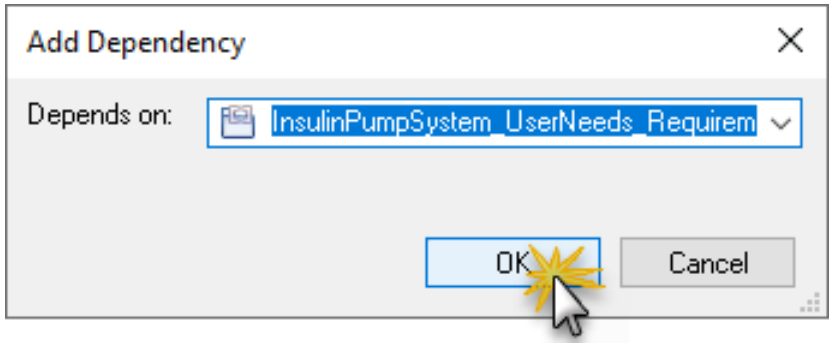


- Choose the **_UserNeeds_RequirementPkg** and click **OK**

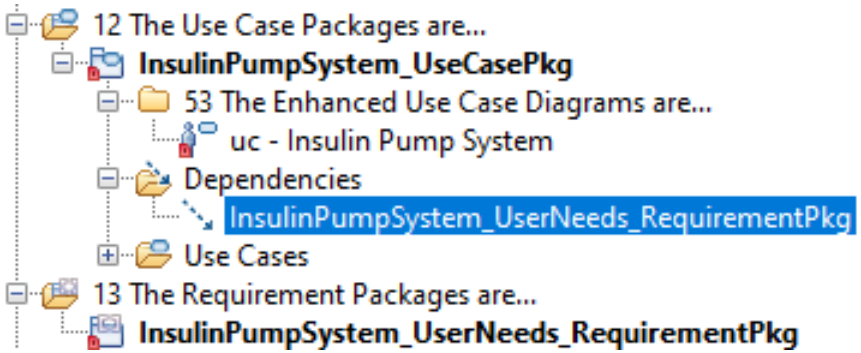


Choose the dependency target package

- Click **OK** to create the dependency relation (and then click in the white space to keep the default name)

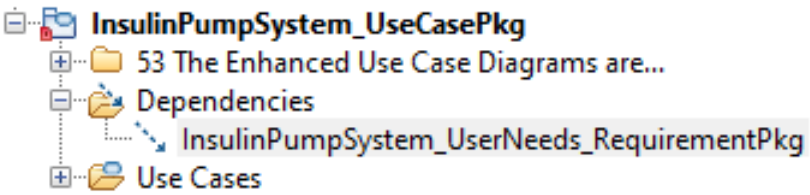


If we don't edit it, then Rhapsody will automatically name the dependency based on the target element. It is always suggested to leave the names of dependency relations unchanged

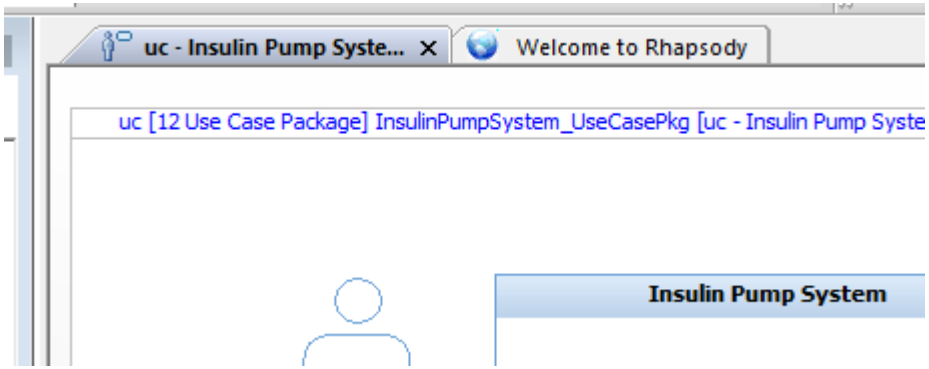


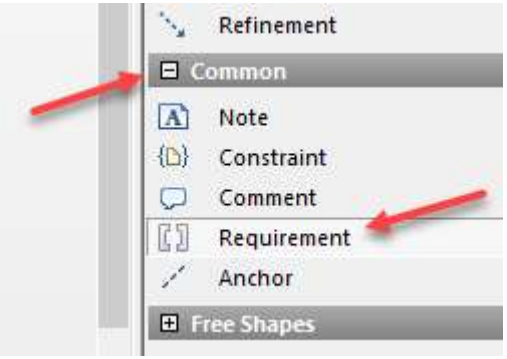
Add a requirement to the diagram

- A dependency from the use case package to the requirements package now exists in the model



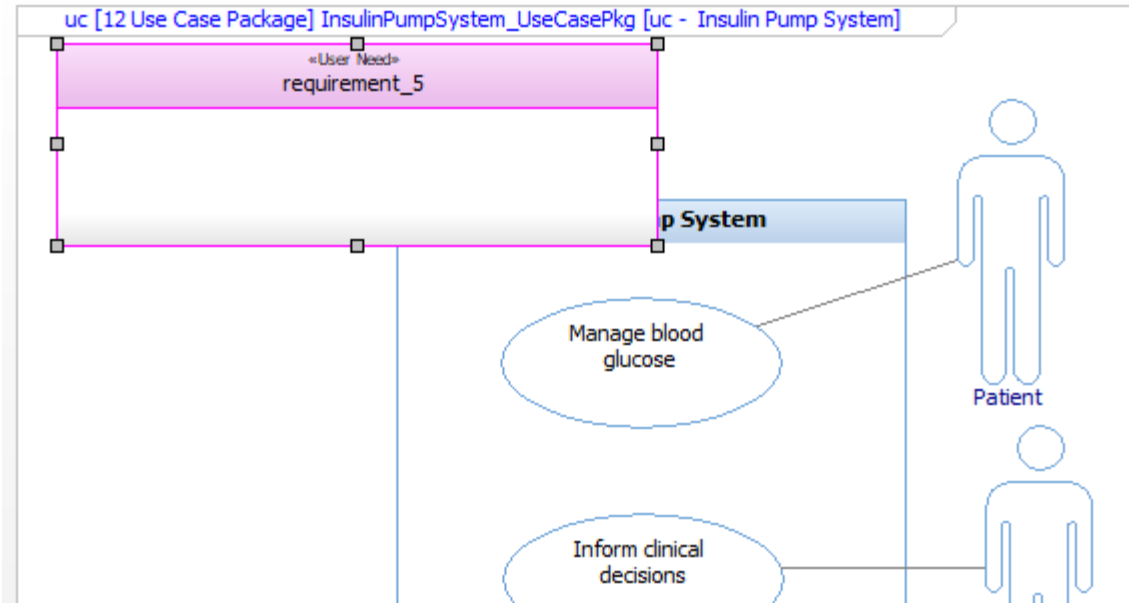
- Locate the **uc – Insulin Pump System** diagram tab and expand the Common section of drawing toolbar to locate the **Requirement** element type

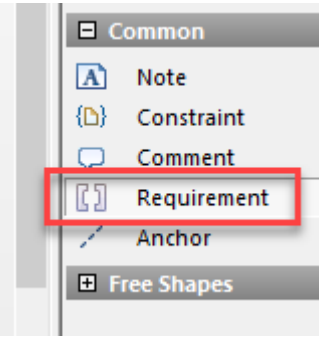




Drop a new requirement on to the diagram

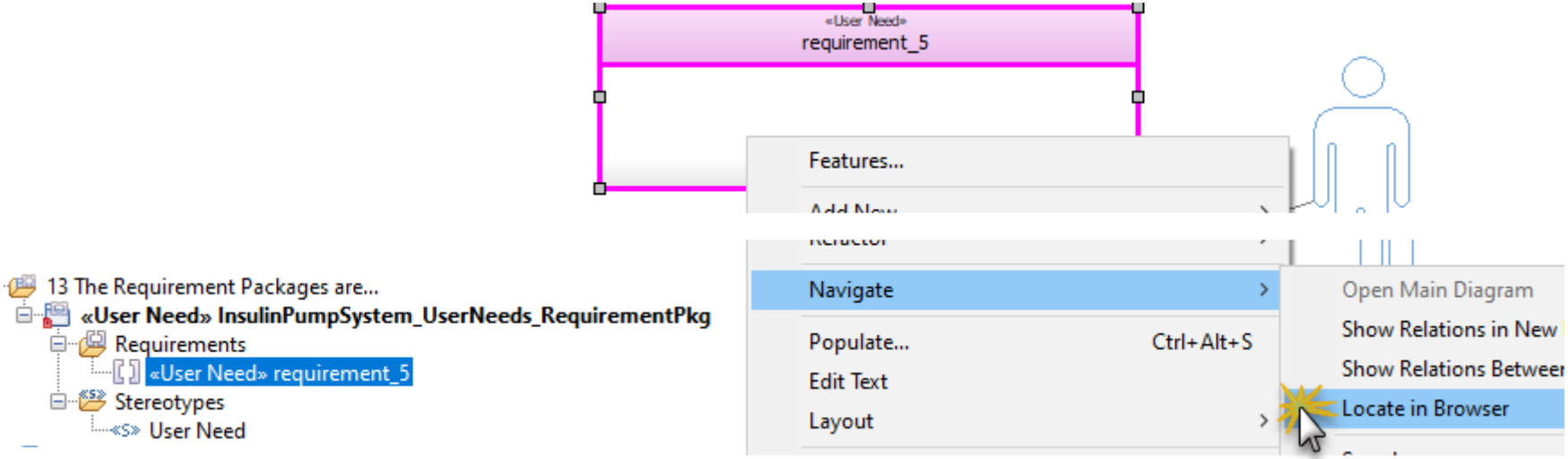
- Drop a new requirement on to the diagram. It is automatically stereotyped as a «User Need» with the formatting applied





View where the requirement is in the browser

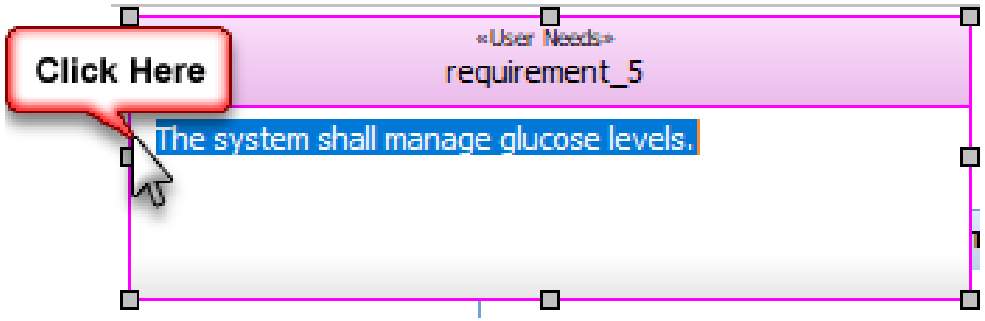
- **Navigate > Locate in browser** (<Ctrl>+L) on the requirement. Due to the dependency added between the packages, the profile has automatically moved it into the requirement package and stereotyped as a **«User Need»**



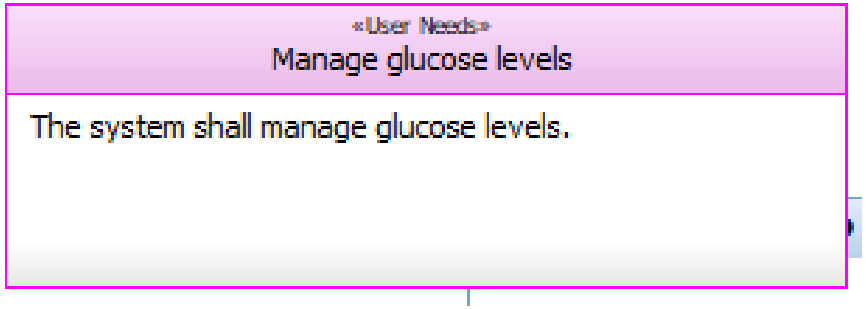
The auto-move functionality is provided by the profile. It helps to have the requirements in a common package to be able to find them, create tables more easily, and synchronize with external sources (e.g., export to .csv for import to DOORS Next)

Change the requirement specification text and name

- Add the specification: **The system shall manage glucose levels.**



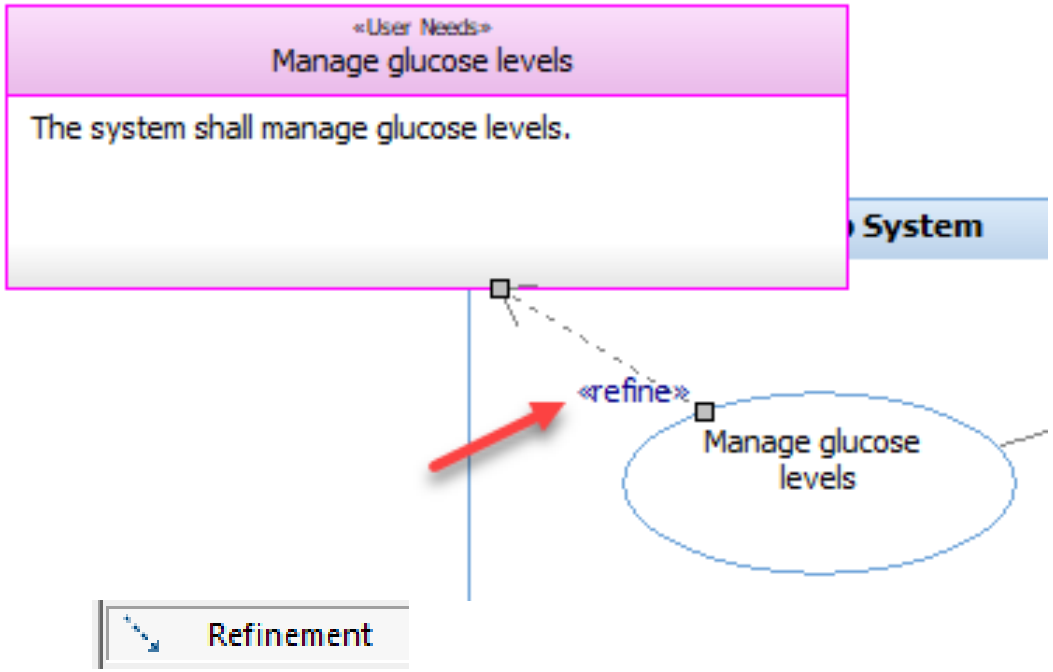
- Change its name to: **Manage glucose levels**



Always give the requirement a unique and meaningful name. This may be replaced with a unique ID once sync is performed with DOORS Next or DOORS classic but having appropriate human readable naming proves very useful until this happens

Draw a Refinement from the use case to the requirement

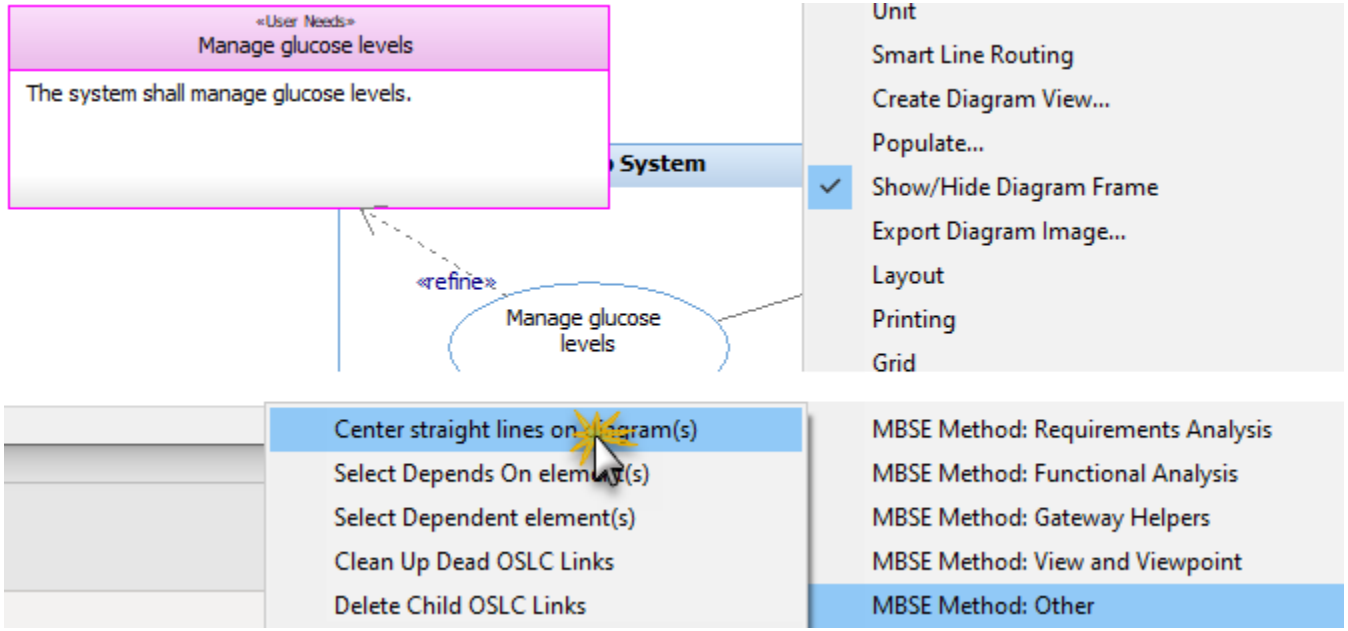
- Draw a **Refinement** relation from the **Manage blood glucose** use case to the **Manage glucose level** requirement



Traceability relations related to requirements always start from a model element and point to a requirement, not from the requirement

Draw a Refinement from the use case to the requirement

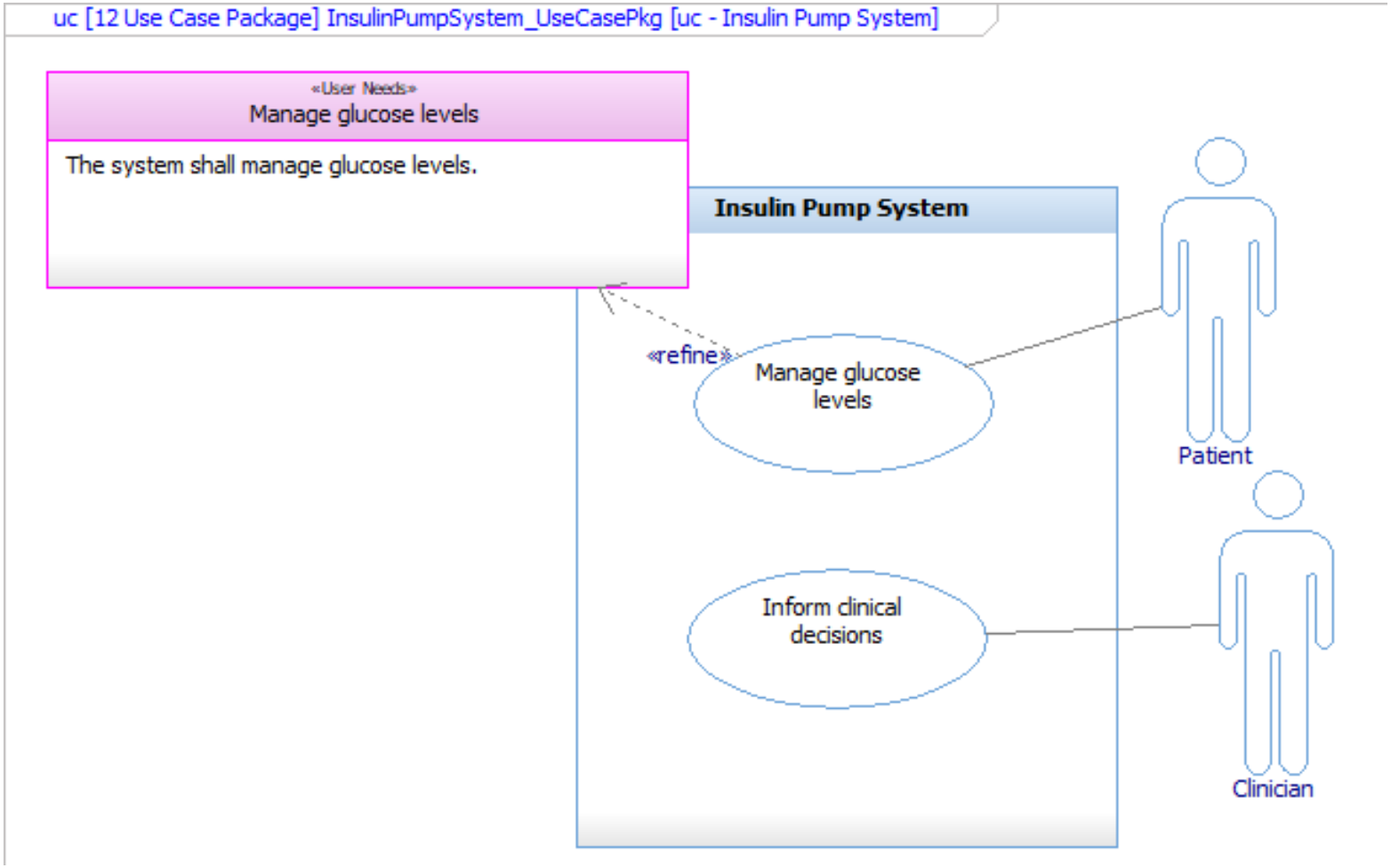
- The profile provides an **MBSE Method: Other > Center straight lines of diagrams(s)** right-click helper for diagrams, that can speed beautifying diagrams. You can try this, if you like (right-click grey area of diagram)



In almost all cases, the profile helper commands will give a dialog that explains what they will do, before they do it. This design philosophy is intended to help users understand what is happening and reduce the risks of new users choosing the wrong option

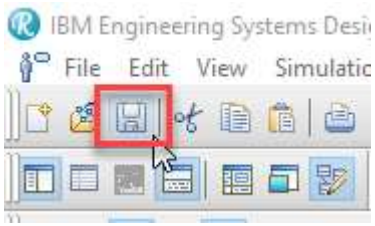
Review the use case diagram

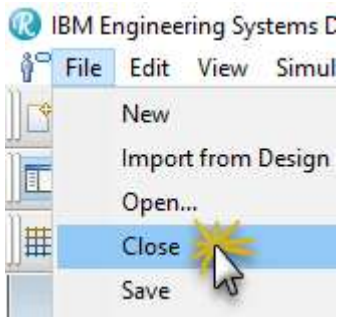
- Check diagram looks something like this



Save and close the project

- The packages we have created so far follow a common pattern which might be repeated for other models of other systems
- The profile provides form-based helpers to create packages automatically to bootstrap a project with a structure
- In the next lab we try the Use Case Package structure wizard in new model
- **Save** this project and **Close** the model





Using **File > Close** rather than **File > Exit** will mean that the project will be available in the Recent Projects list in the File menu

AUTO-CREATE USE CASE PACKAGE STRUCTURE

LAB IA4

"INSULIN PUMP" CASE STUDY

LAB IA4

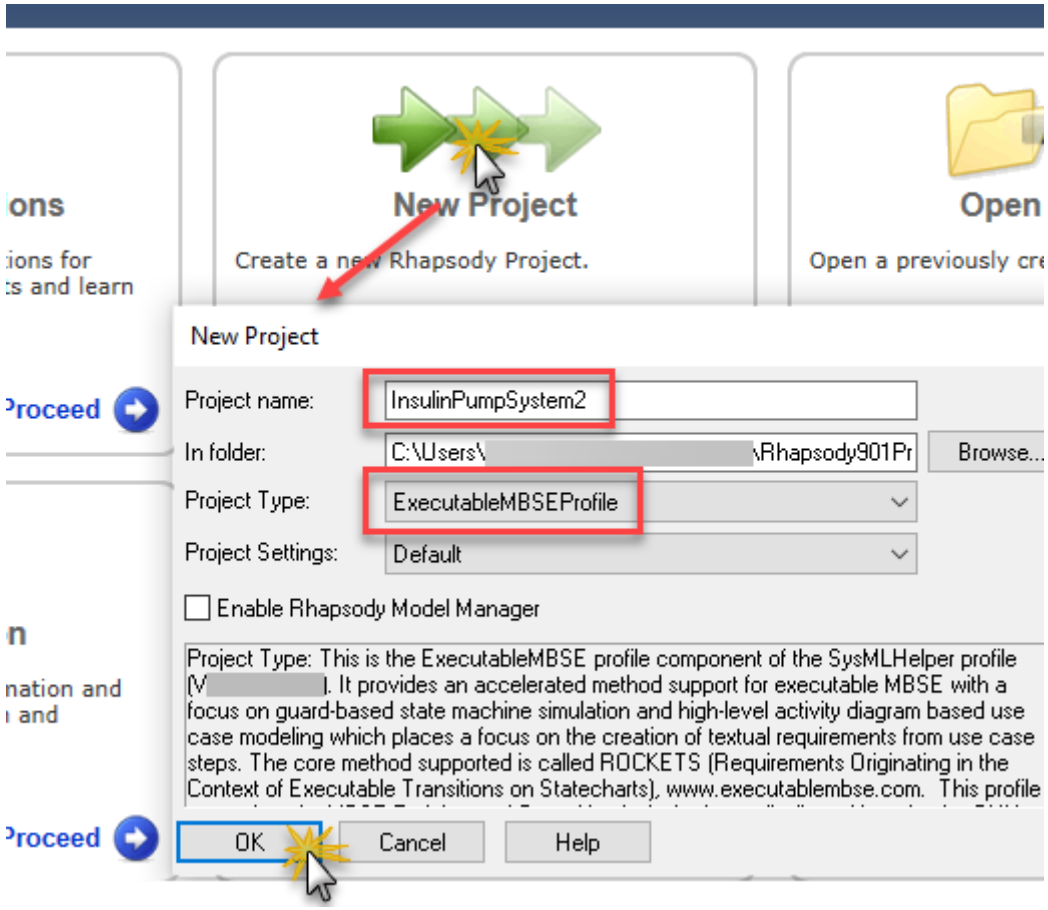
"hear and I forget. I see and I remember. I do and I understand" (Confucius 551 BC - 479 BC)

Introduction to the lab

- In this lab we use the Use Case Package structure… menu to create a combination of the following three different package types:
  - **Use case package:** A container for use cases and use case diagrams
  - **Actor package:** A container for actors shared by different use case packages
  - **Requirement package:** A container for requirements shared by different packages, with additional tables designed to view requirements and their traceability
- The helper also automatically adds a dependency from the use case package to the requirements package, and can automatically create the stereotype used to differentiate different requirement types

Create a New Project

- Create a **New Project** called **InsulinPumpSystem2** in a Read/Write folder using the **ExecutableMBSEProfile** as the **Project Type**



View the Log window

- View the **Log** window & Ensure the profile loaded

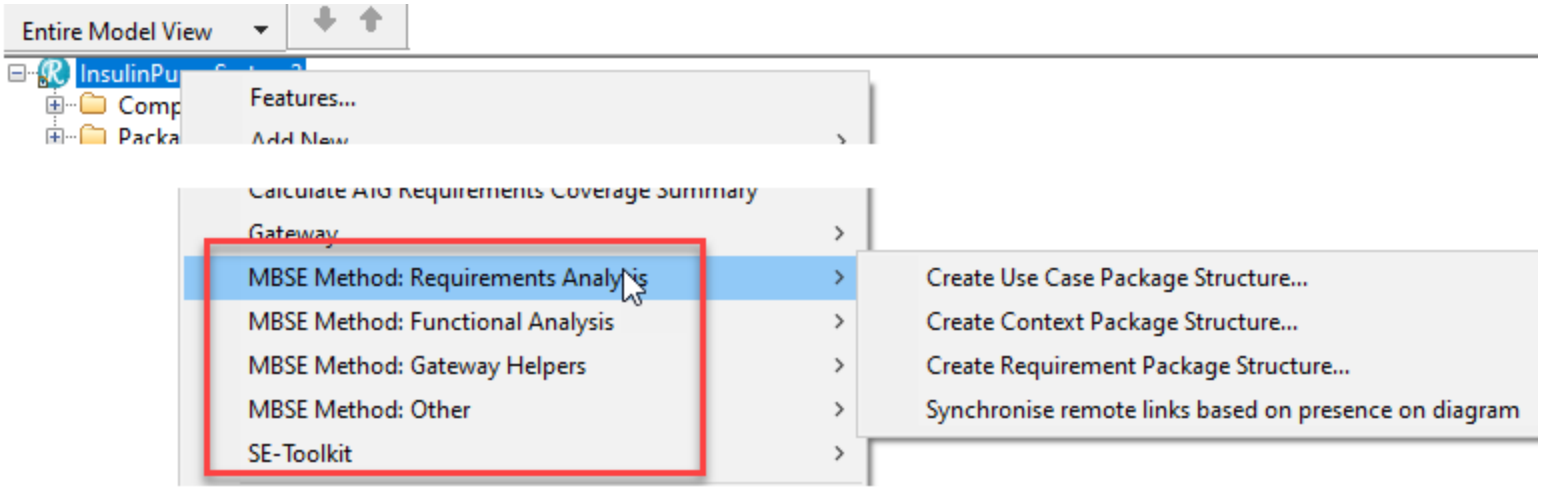
```
SysMLHelperPlugin is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even
New right-click 'MBSE Method' commands have been added.
Info   : ExecutableMBSE_RPApplicationListener was loaded - Listening for events (double-click etc)
Info   : The ExecutableMBSE profile version is
Info   : The project called InsulinPumpSystem does not have the ProfileDate and ProfileVersion tags set
Your installed ExecutableMBSE profile version is
It is recommended to right-click and run the Profile's Setup project properties command
to update the project to the latest profile before proceeding
```

Log | Check Model | Build | Configuration Management | Animation

The ExecutableMBSE profile includes a Java plugin that adds new menus to assist with method automation. The Log window contains an indication that this plugin has loaded plus which version it is. If you don't see this, then it's probable your $OMROOT/Profiles/SysMLProfile folder is not in the correct location! This must be fixed before proceeding

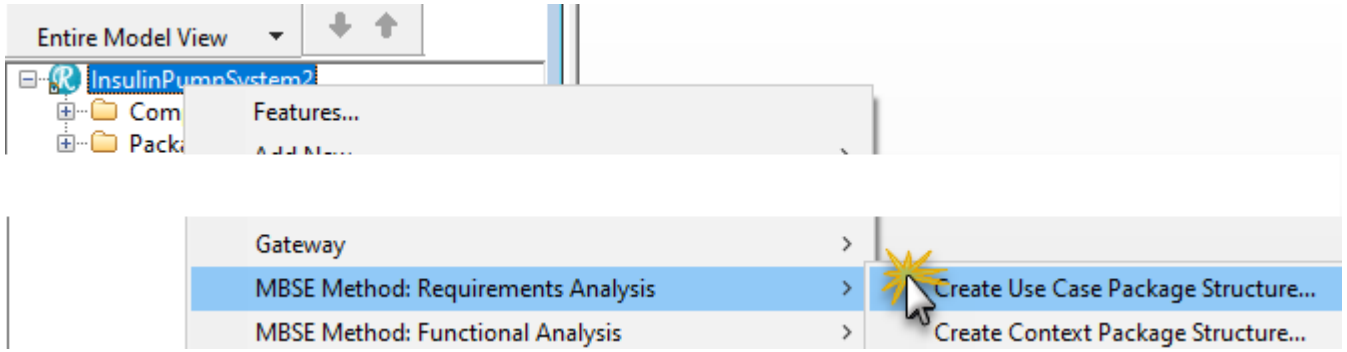
View the project right-click menu

- Right-click the project and view the context menu. Down the bottom of this are menus added by the Executable MBSE profile hep file that begin with MBSE Method: <Group Name>

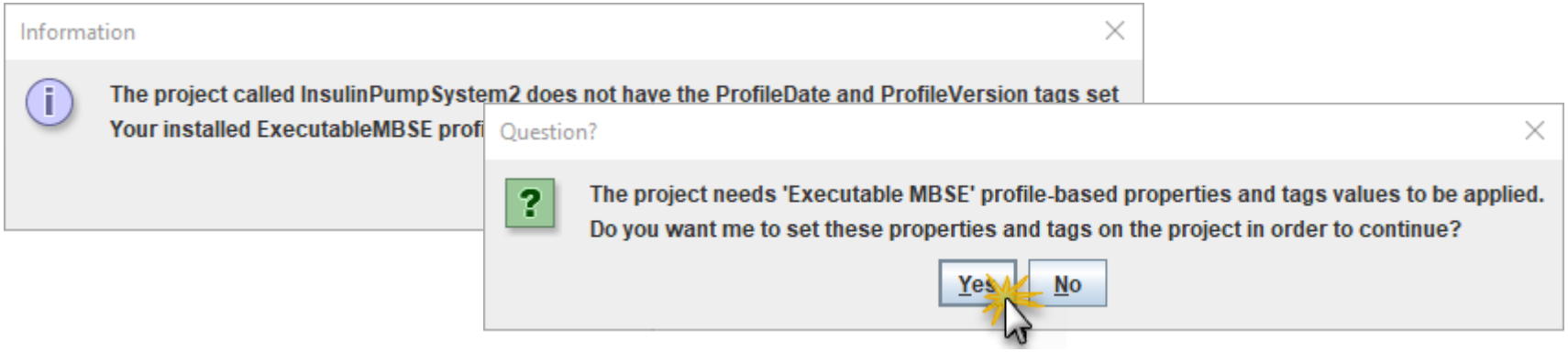


Create Use Case Package Structure…

- The most common method to start a project with is **MBSE Method: Requirements Analysis > Create Use Case Package Structure…**
- Try this now

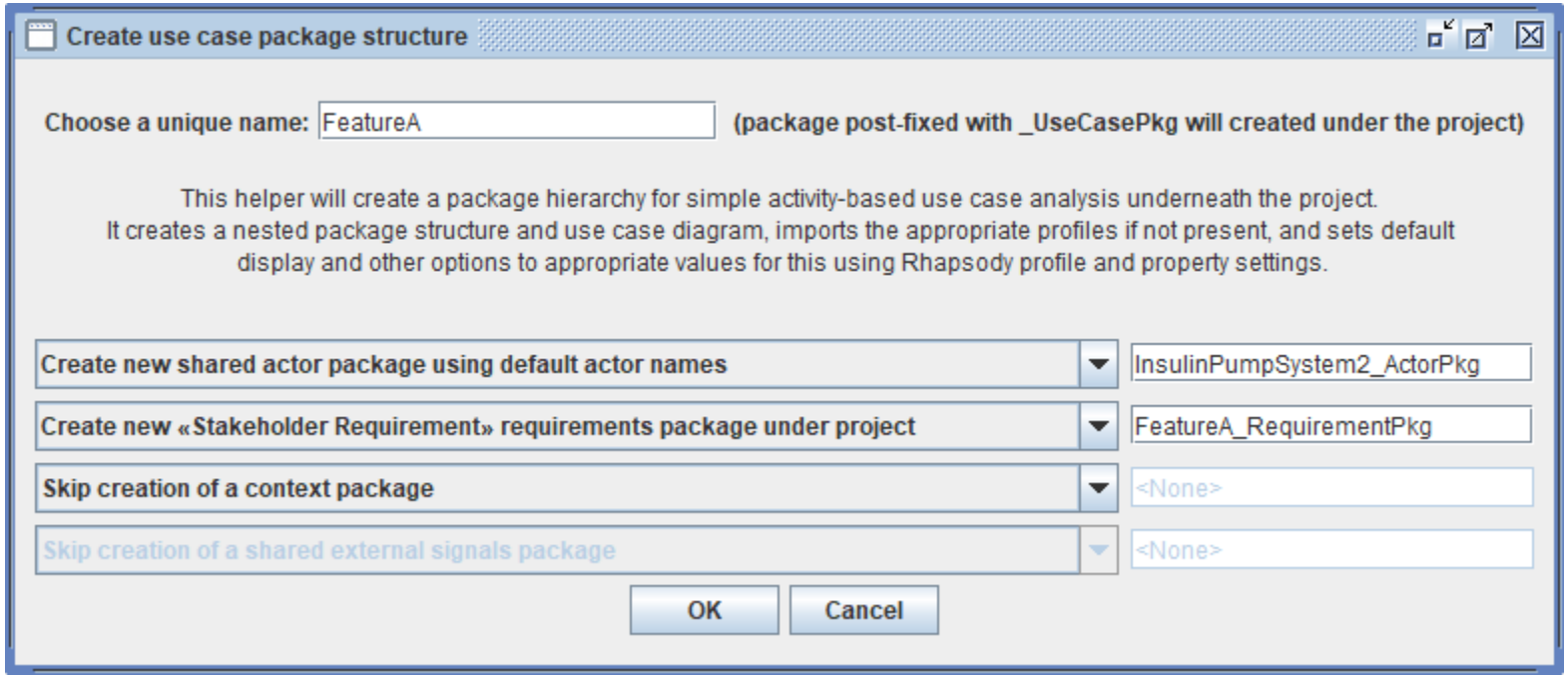


- Click **OK** to proceed and **Yes** to setup the project properties



View the dialog

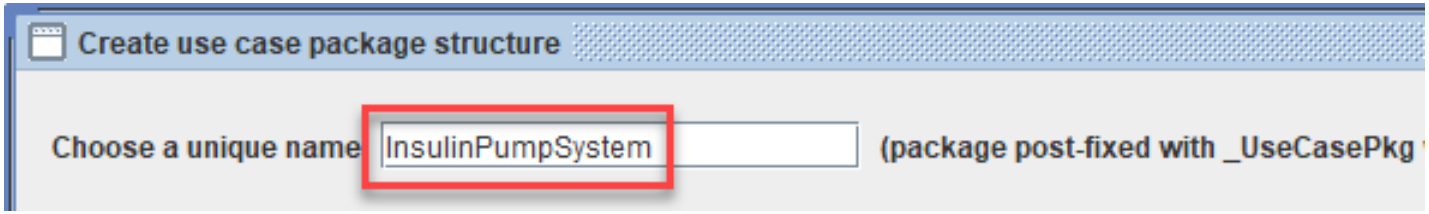
- A dialog pops up that has been provided by the profile



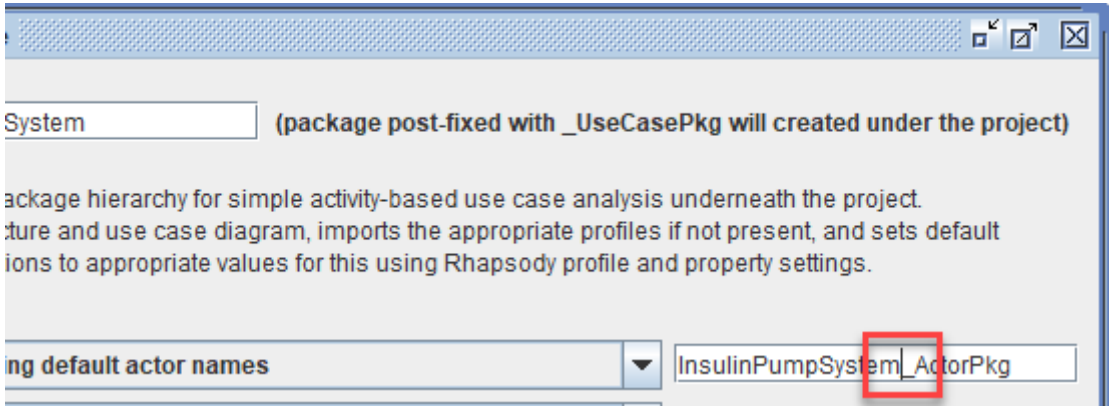
The options here are setup to create a use case package, requirement package and actor package at the same time

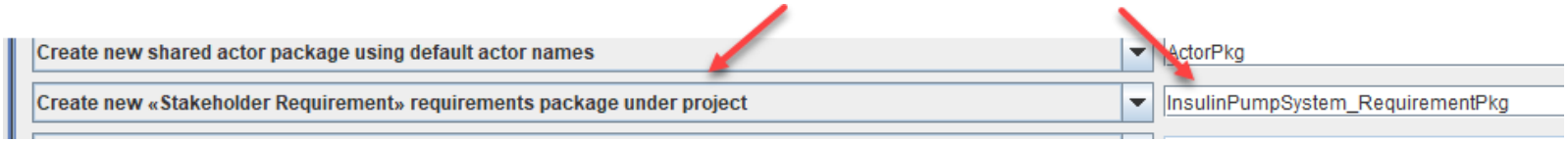
Choose the unique name InsulinPumpSystem

- Choose the unique name **InsulinPumpSystem**



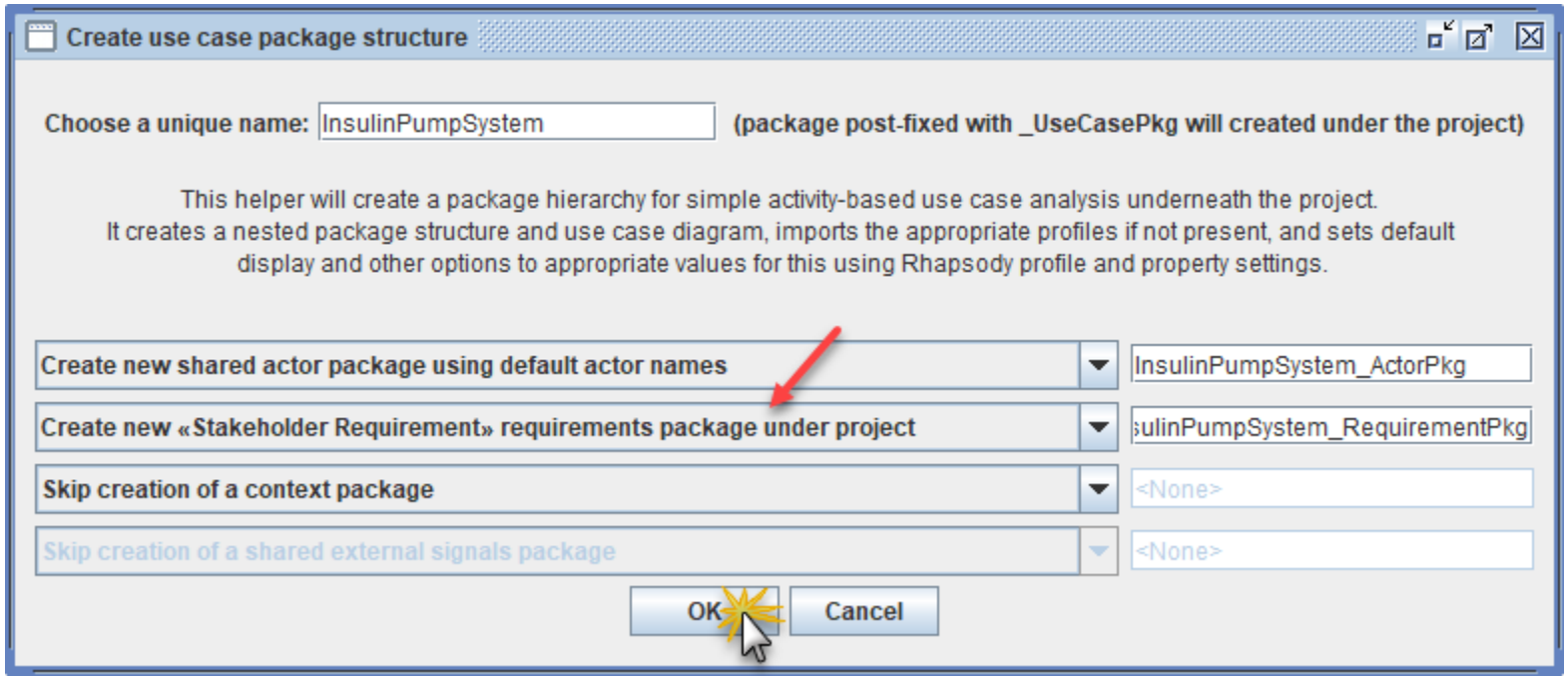
- Remove the 2 from the Actor package name:





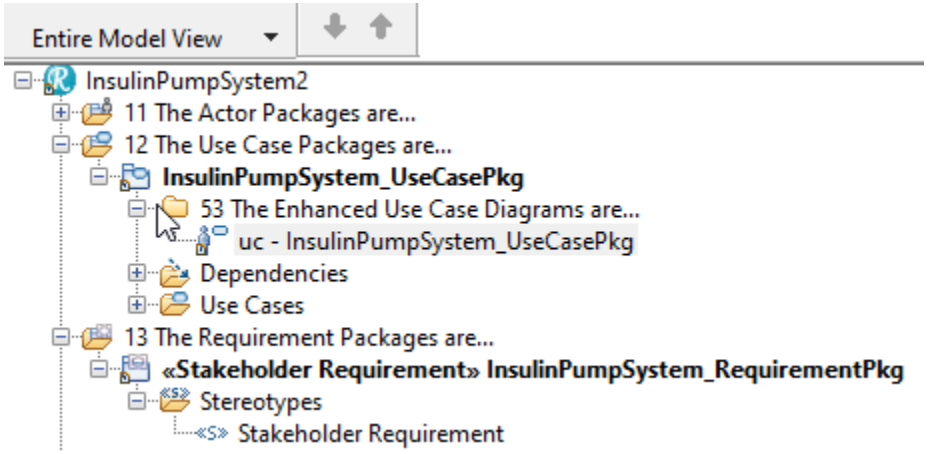
Check settings and click OK

- The option to create a stereotyped requirements package has been selected by default. Check the settings below
- Click **OK** to create the model

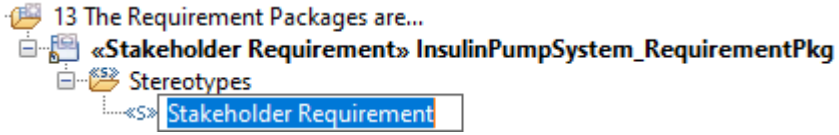


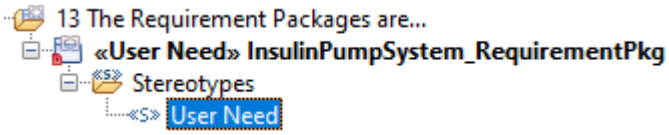
View the created structure

- The packages are automatically created including the dependency from the use case package to the requirements package



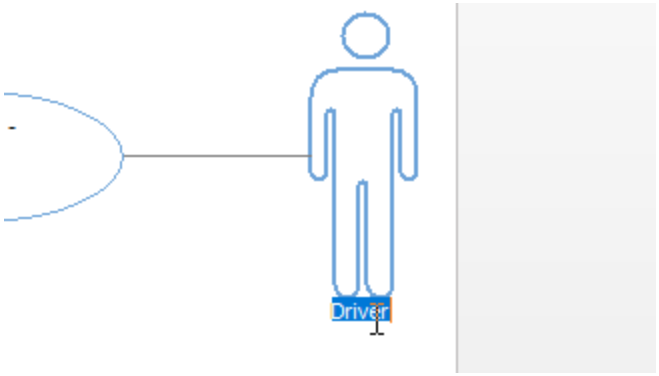
- Click the **Stakeholder Requirement** stereotype and rename it to **User Need**

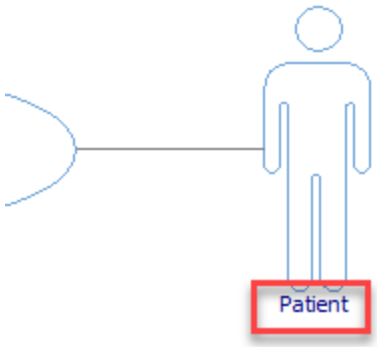




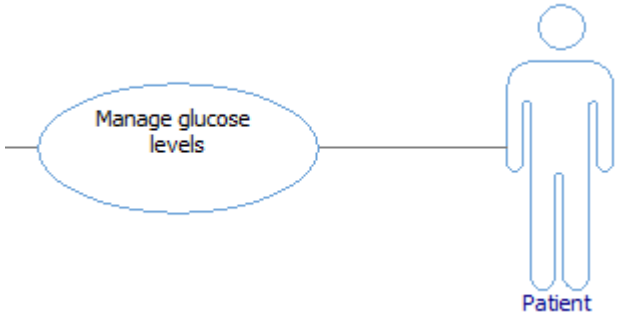
Rename the Driver actor to Patient

- Double-click the **Driver** actor's name and change it to **Patient**





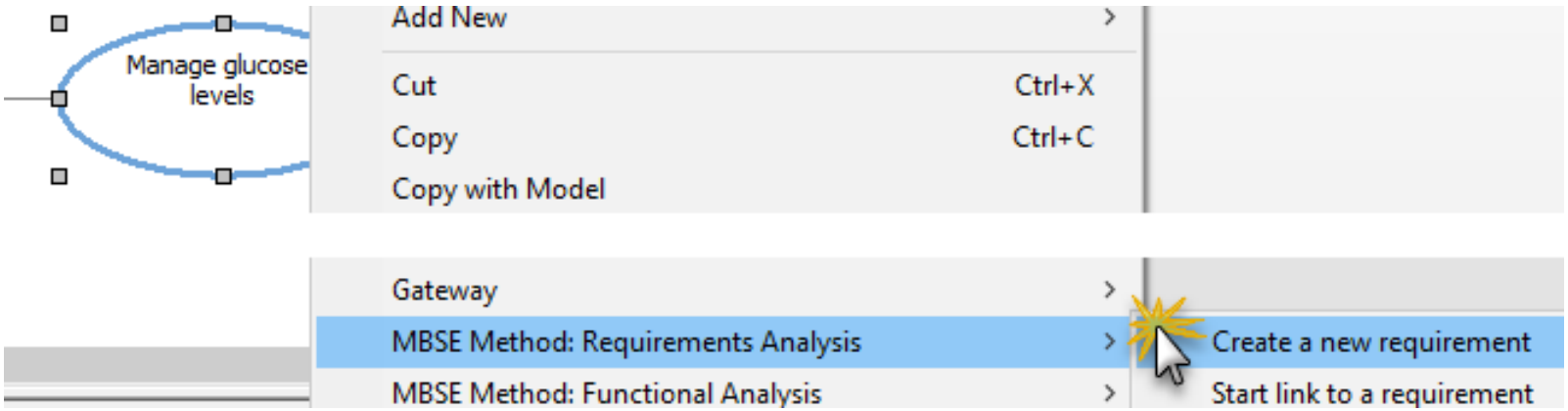
- Single-click the text to rename the use case to: **Manage glucose levels**



Double-click functionality is overridden on a use case. If a dialog pops up asking to create an activity diagram, then click No.

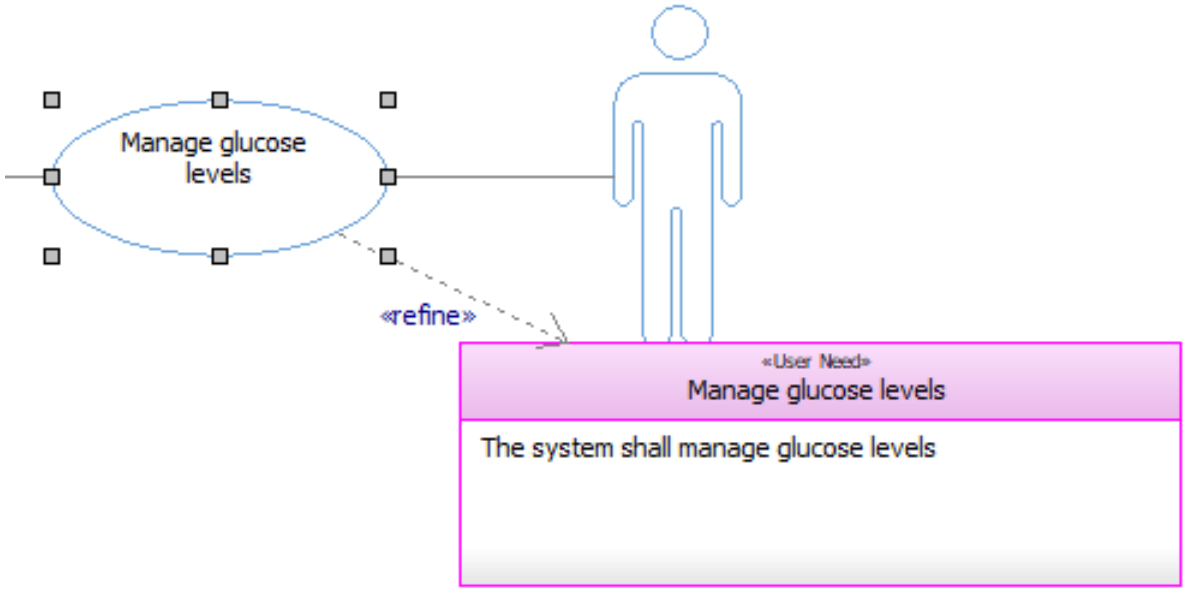
Create a new requirement

- A simple helper can be used to create a requirement and associated traceability at the same time
- Right-click **Manage glucose levels** and choose **MBSE Method: Requirements Analysis > Create a new requirement**



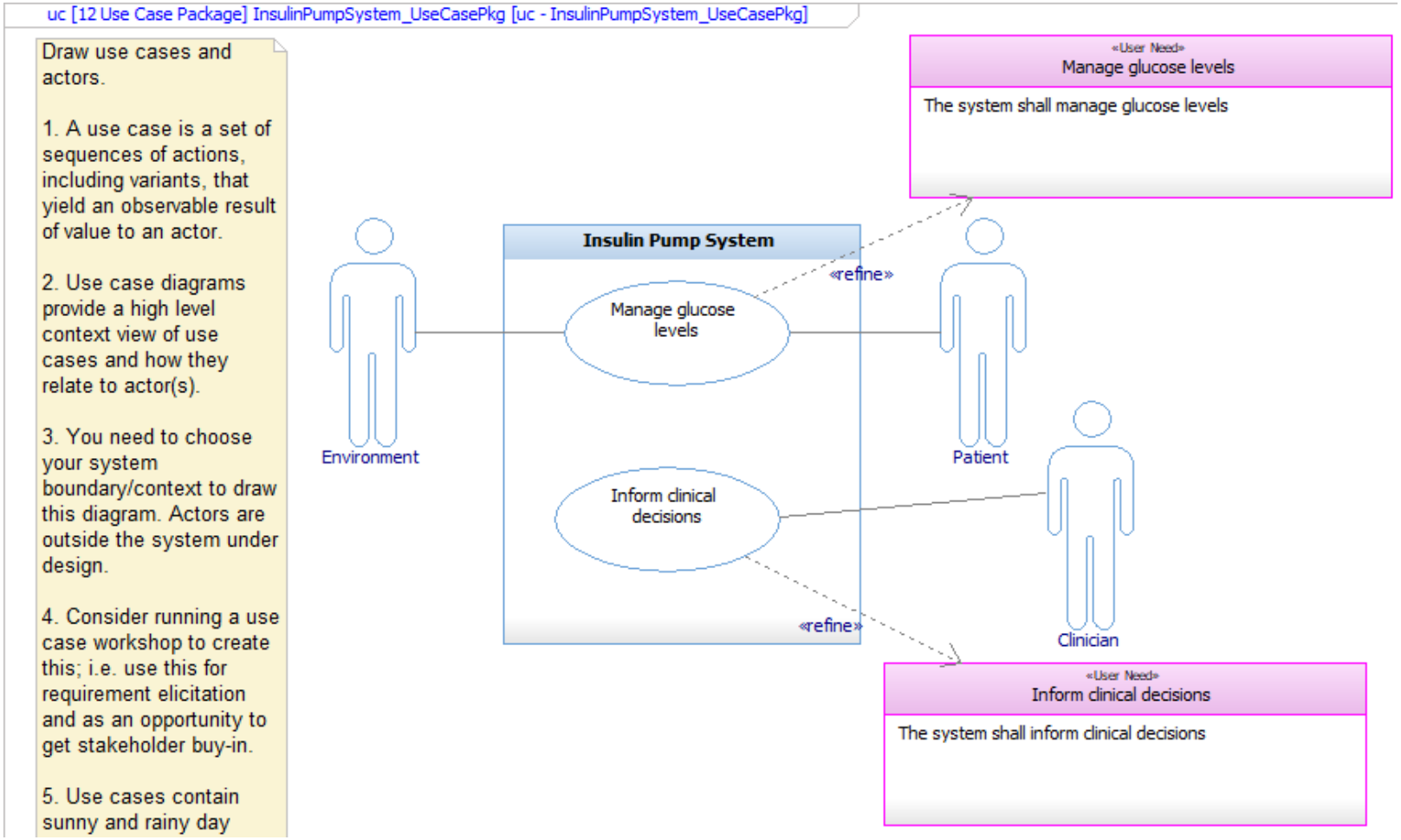
Create a new requirement

- The helper has created the requirement, established a refinement relationship to it, and populated the requirement on the diagram. Some initial text has been added based on the source element



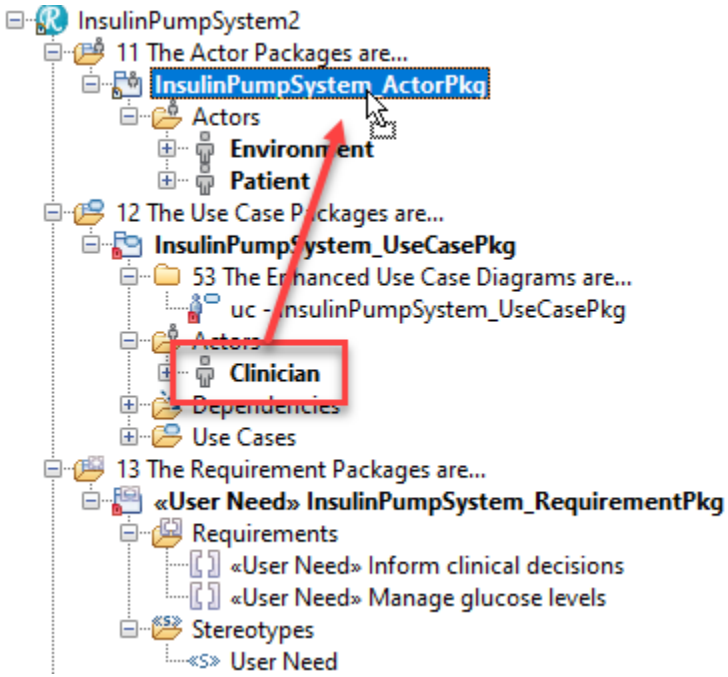
Finish the diagram

- Add the **Inform clinical decisions** use case and **Clinician** actor to finish the diagram below

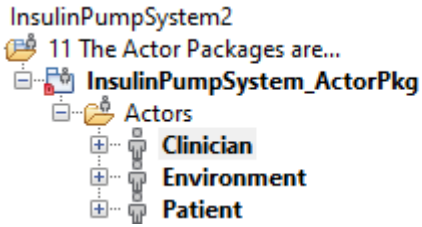


Move the Clinician into the Actor package

- Move the **Clinician** into the **_Actor** package



- Check the package structure looks the same as this:



It was much quicker it was to create this model structure using the helper. You can get straight into use case and requirement analysis

REQUIRMENTS AND TABLES

LAB IA5

"INSULIN PUMP" CASE STUDY

LAB IA5

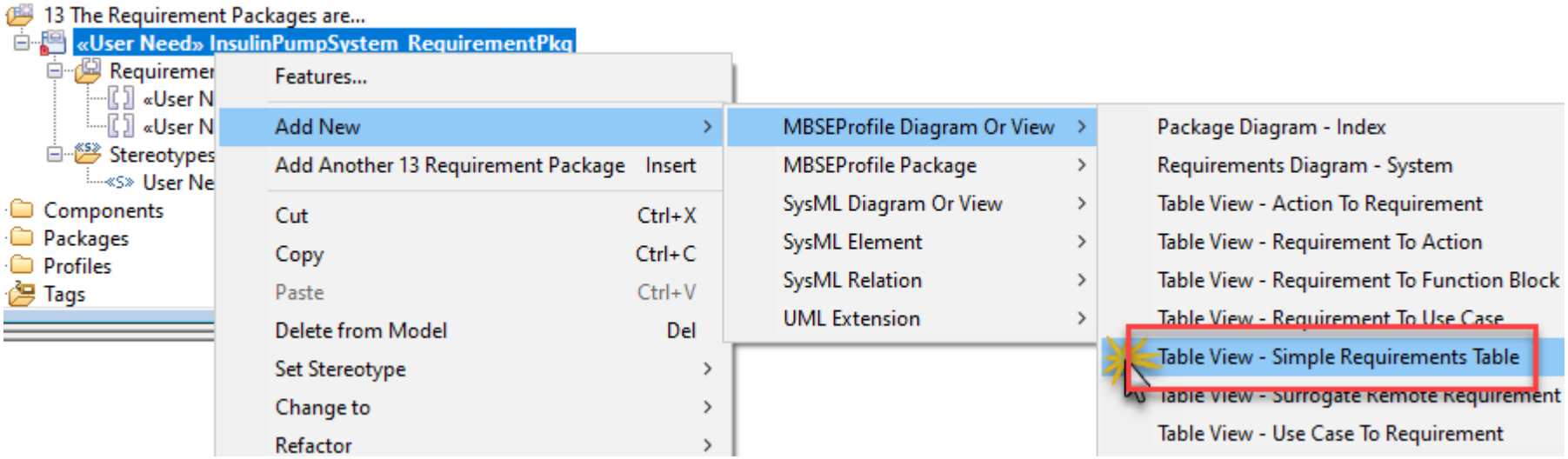
"hear and I forget. I see and I remember. I do and I understand" (Confucius 551 BC - 479 BC)

Requirements and Tables

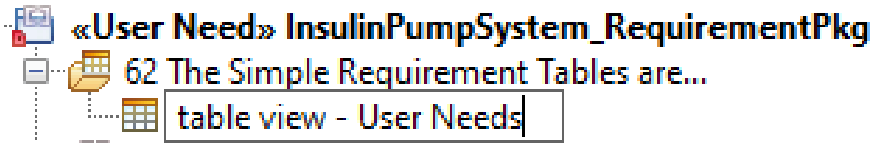
- Requirements and their associated traceability are often best shown in table views. The profile adds several default layouts that can be useful
- It makes use of new term **Table View** stereotypes, that automatically apply a given **Table Layout** when the view is created. This speeds up creating tables based on these layouts and means that different views can be added to different new term package types, e.g., subsystem table is appropriate in the system architecture package, rather than the use case package

Add a TableL – Simple Requirements table

- Right-click the requirement package and choose **Add New > MBSEProfile Diagram Or View > Table View – Simple Requirements table**

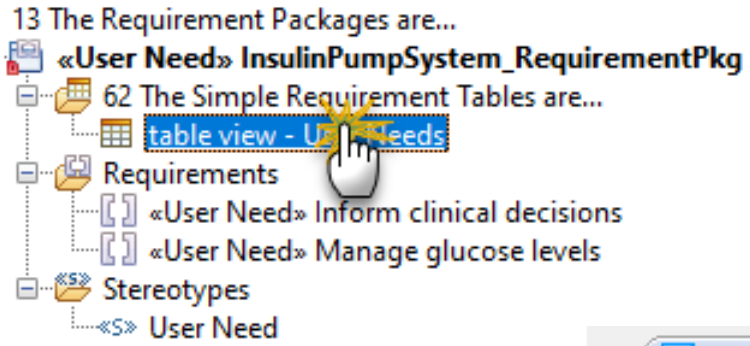


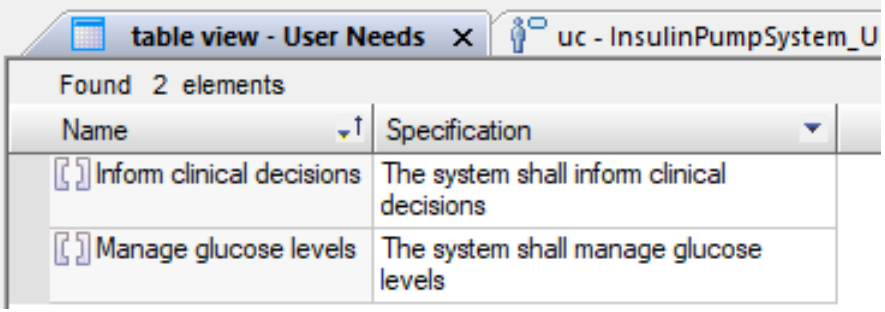
- Call it **table view – User Needs**



Double-click to render the table

- Double-click the table view to render it

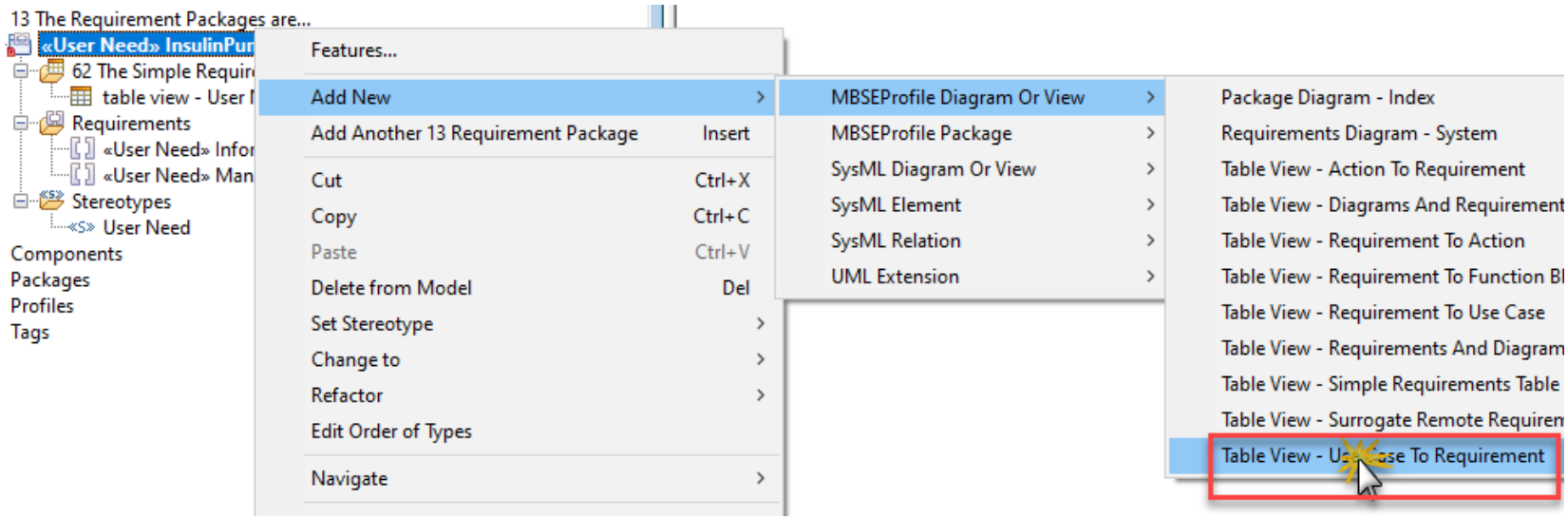




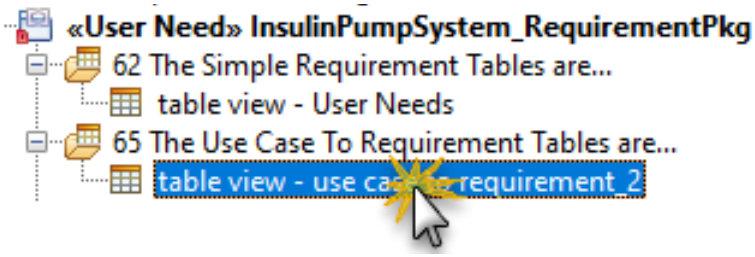
The profile listener automatically scopes the table to the requirement package when it's created, hence we don't need to do this

Add a Table View – Use Case To Requirement

- Right-click the requirement package and choose **Add New > MBSEProfile View > Table View – Use Case To Requirement**



- Double-click to render the table



View the relation table

- A relation table based on a context pattern is shown

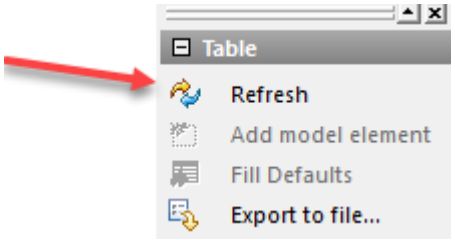
table view - use case to r... | table view - User Needs | uc - InsulinPumpSystem_Us... | Welcome to Rhap

Found 2 elements

Use Case	Requirement	Specification
Inform clinical decisions	Inform clinical decisions	The system shall inform clinical decisions
Manage glucose levels	Manage glucose levels	The system shall manage glucose levels

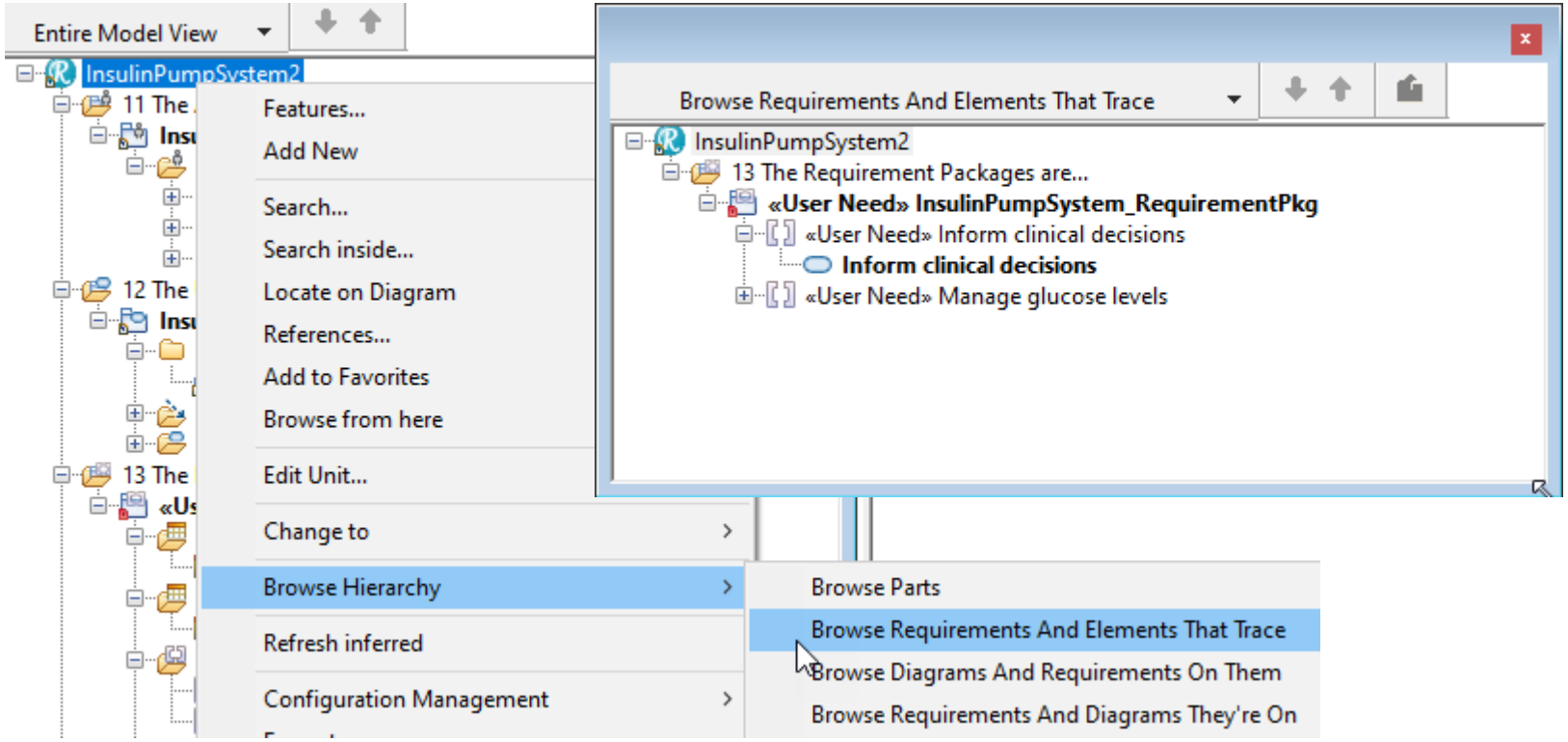
This is showing that the use case manage blood glucose has a refinement dependency to the manage glucose level requirement

- For Info: If you added new requirements and refinement relations to the use case diagram, you could update this table by clicking **Refresh** in the drawing toolbar



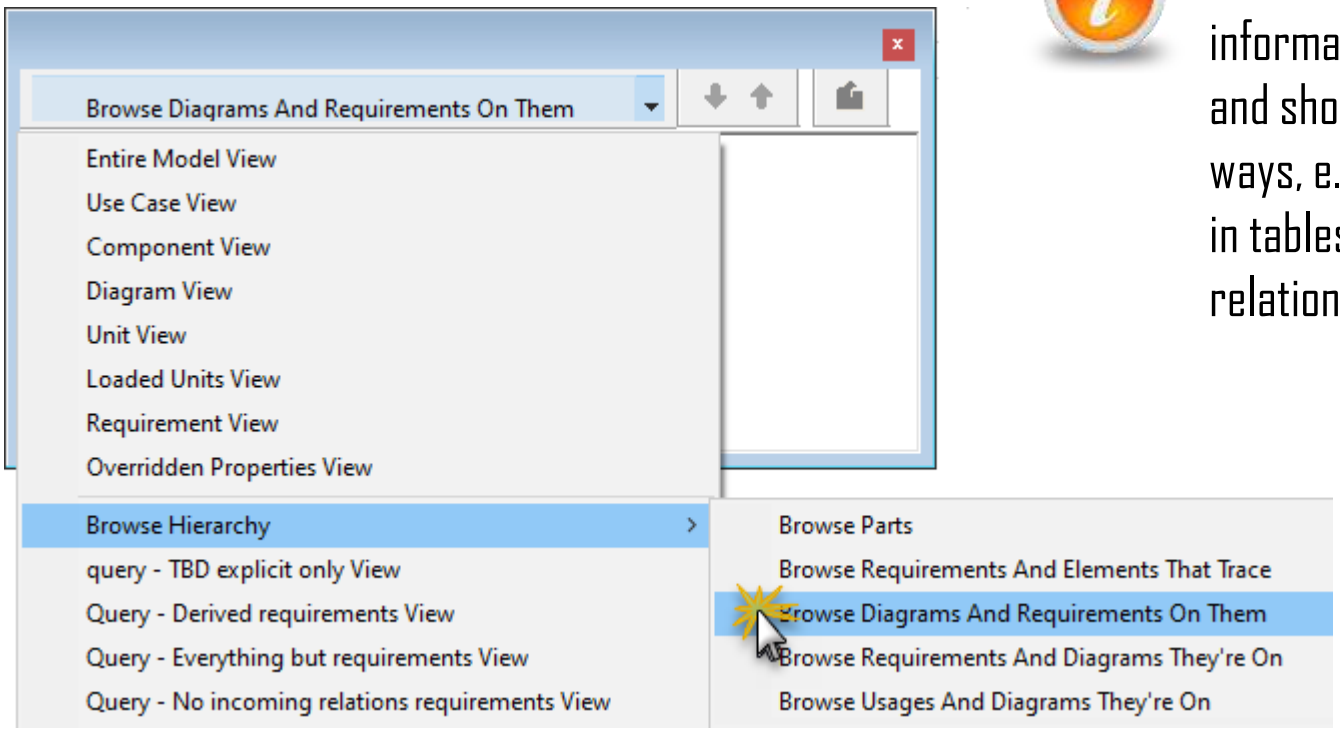
Browse Hierarchy > Browse Traceability

- Optional Extension: The profile also adds a Browse Traceability window for viewing requirement traceability
- Right-click the project and choose **Browse Hierarchy > Browse Requirements and Elements That Trace**. A new window pops up



View the traceability browser

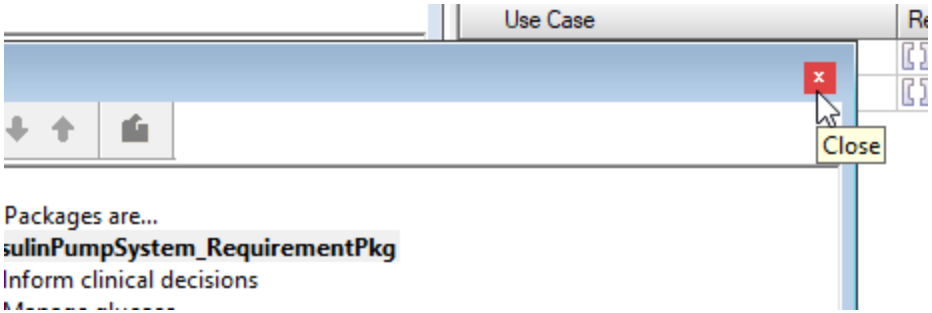
- Try switching new browser to **Browse Hierarchy > Browse Diagrams and Requirements On Them**



With models the same information can be viewed and shown in different ways, e.g., on diagrams and in tables and matrices, and relationship browsers

View the traceability browser

- Close the Browser Traceability window when you're finished



INSTALLING THE SYSML HELPER

Appendix A

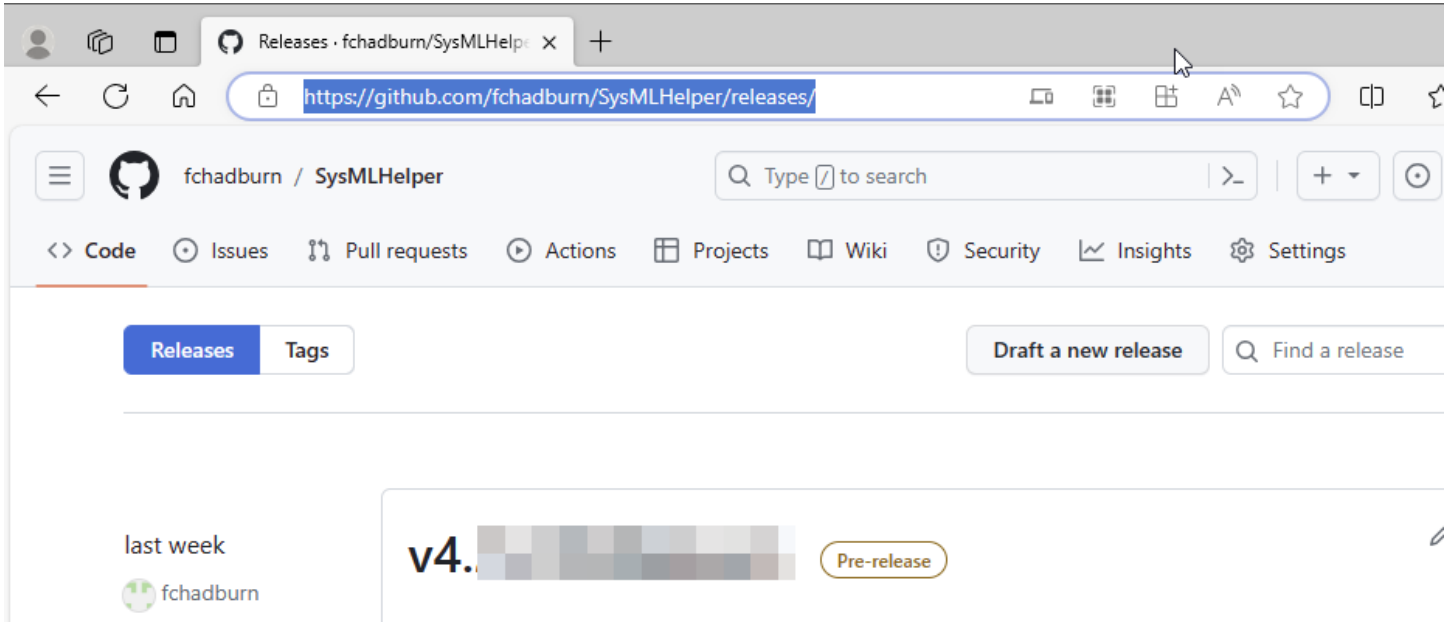
"INSULIN PUMP" CASE STUDY

APPENDIX A

"I hear and I forget. I see and I remember. I do and I understand" (Confucius 551 BC - 479 BC)

Obtain the profile from Github

- Go to https://github.com/fchadburn/SysMLHelper/releases/ to find the releases. Locate the version needed

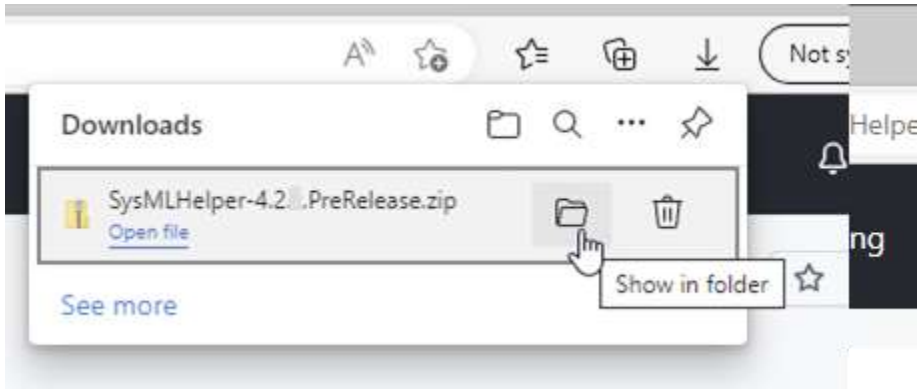


You can also visit my site at: http://www.executablembse.com/ which contains some further info on latest changes

Obtain profile from Github

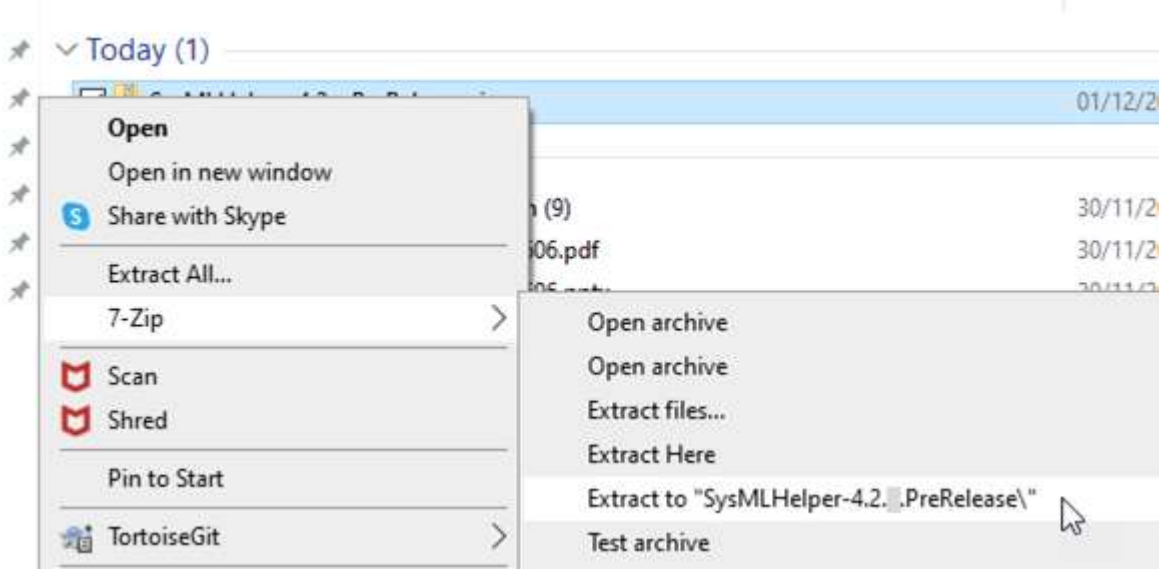
- Click to download Source code (zip)

- Locate file

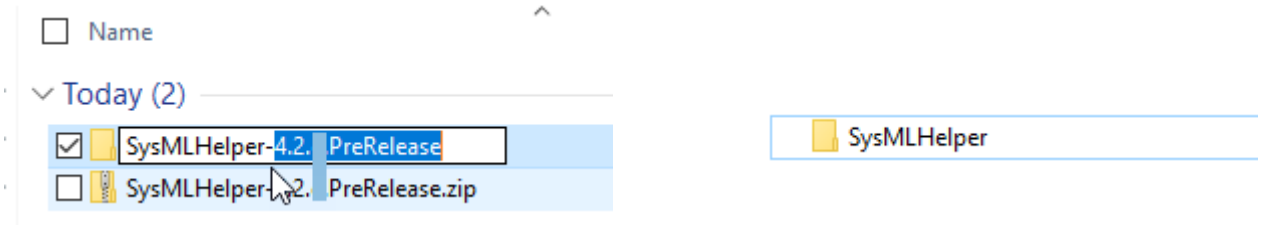


Extract to folder of same name

- Extract to folder of same name

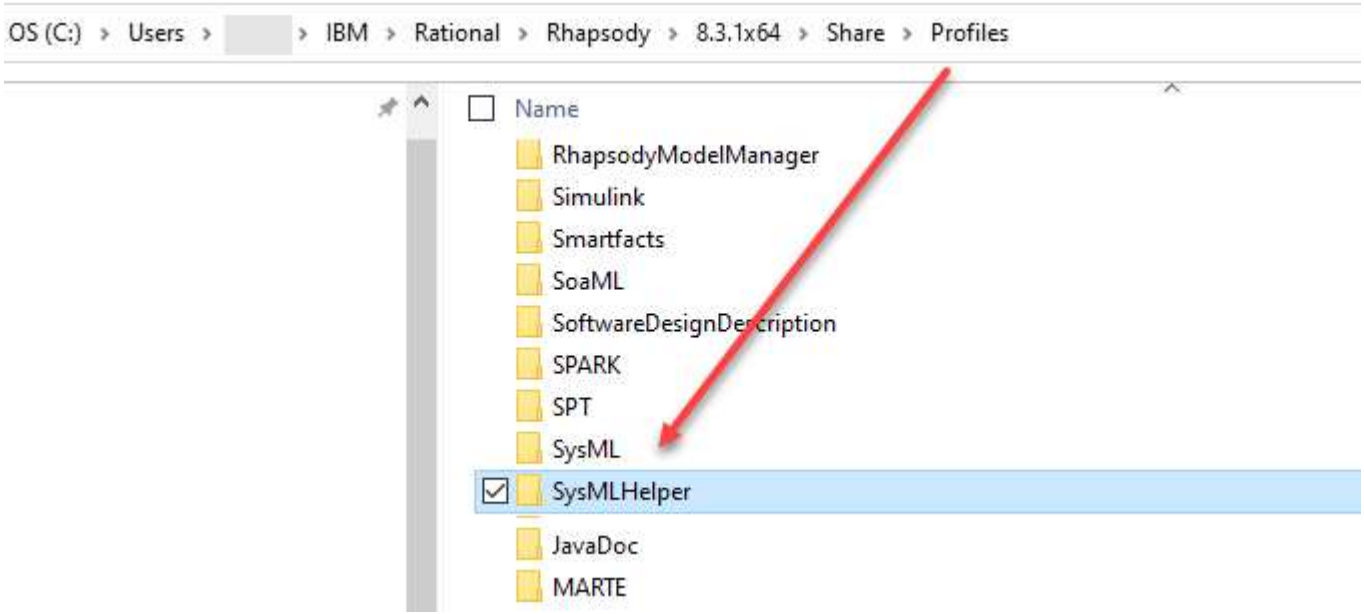


- Remove the release part of name so top-level is called SysMLHelper



Locate your Rhapsody share/profiles folder

- This is where Rhapsody's factory (out-of-the-box) profiles are located. E.g. in 9.0.1. this is C:\Program Files\IBM\Rhapsody\9.0.1\Share\Profiles
- This is where the SysMLHelper profile needs to go, e.g., 8.3.1x64 may be



- Delete, or move out, rather than rename the existing profile you have or overwrite over the top (this is very important as Rhapsody assumes only one .sbs/.sbsx file of a particular name exists under its Profiles folder)