# C:\Users\Johnny El-Hayek\Desktop\Fall 2015\ECSE 429\Project\Dependency Graph - Part 1.PNGC:\Users\Johnny El-Hayek\Desktop\Fall 2015\ECSE 429\Project\Dependency Graph - Part 2.PNGDependency Graph

# Integration Strategy

Test the 15 classes using Bottom-up integration testing in the following order:

1. Test:
   1. AssignToHandler (driver for AssignToHandler needed)
   2. MessageController (driver for MessageController needed)
   3. MessageViewController (driver for MessageViewController needed)
   4. ValueSpecificationHandler (driver for ValueSpecificationHandler needed)
2. Test:
   1. MessageAssignToHandler + AssignToHandler + MessageController (driver for MessageAssignToHandler needed)
   2. ReferenceValueHandler + MessageController (driver for ReferenceValueHandler needed)
   3. MessageHandler + MessageViewController (driver for MessageHandler needed)
   4. LifelineViewHandler + MessageViewController (driver for LifelineViewHandler needed)
   5. FragmentsController + MessageViewController (driver for FragmentsController needed)
3. Test:
   1. MessageViewHandler + FragmentsController + MessageViewController (driver for MessageViewHandler needed)
   2. AssignmentStatementHandler + FragmentsController + MessageViewController (driver for AssignmentStatementHandler needed)
   3. AssignmentAssignToHandler + FragmentsController + MessageViewController (driver for AssignmentAssignToHandler needed)
   4. CombinedFragmentHandler + FragmentsController + MessageViewController (driver for CombinedFragmentHandler needed)
   5. ExecutionStatementHandler + ValueSpecificationHandler + FragmentsController + MessageViewController (driver for ExecutionStatementHandler needed)
   6. InteractionConstraintHandler + ValueSpecificationHandler + FragmentsController + MessageViewController (driver for InteractionConstraintHandler needed)
4. Test everything together (drivers for MessageAssignToHandler, ReferenceValueHandler, MessageHandler, LifelineViewHandler, MessageViewHandler, AssignmentStatementHandler, AssignmentAssignToHandler, CombinedFragmentHandler, ExecutionStatementHandler, and InteractionConstraintHandler needed)

# Justification of Chosen Integration Strategy

We choose bottom-up integration because we do not need to implement any stubs to perform integration testing compared to other integration testing strategies. While we would need to implement more drivers, drivers are usually simpler to implement than stubs, so this is still advantageous. In addition, the reusable components (i.e. the lower level components) are tested more thoroughly.

If we choose the top-down approach, we would need to implement stubs for every test (except the ones which cover an entire dependency branch). This would require more time to complete the testing. Of course, we would not need to worry about drivers as much as bottom-up integration. Also, the reusable components would not be adequately tested since they would be tested least. However, major design flaws related to higher level components would be found first in top-down integration. If we choose sandwich integration (which is a combination of top-down and bottom-up integration), we would also have to implement stubs for some tests, and this adds complexity that we want to avoid. If we choose big bang integration, it would not be easy to isolate faults in such a large system, especially faults related to interfaces between components. Finally, we cannot choose risk-driven integration since we do not accurately know the criticality of each of the components of the system.

(Of course, we took into account the work we would have to put in for part 3 when deciding on this strategy, and we agreed that bottom-up integration would be the most adequate for us.)

# Description of drivers, mocks, stubs

Our testing strategy does not require any mocks or stubs for the three classes we have to test. We only require drivers for MessageHandler, MessageViewHandler, and MessageViewController. These are simply the JUnit test methods that we will write for part 3 that will exercise these three tested components.

MessageViewController can be tested without stubs or mocks since it does not depend on any of the other 14 classes listed in this part of the project. MessageHandler and MessageViewHandler can be tested without stubs or mocks since they only depend on components which will have already been tested at that point (i.e. MessageViewController and FragmentsController) since we are using bottom-up integration.