Security Assurance Case Design Tool

...

Dec1718 - Brandon Huegli, John Koehn, Jordan Lawrence
Client/Advisor:...

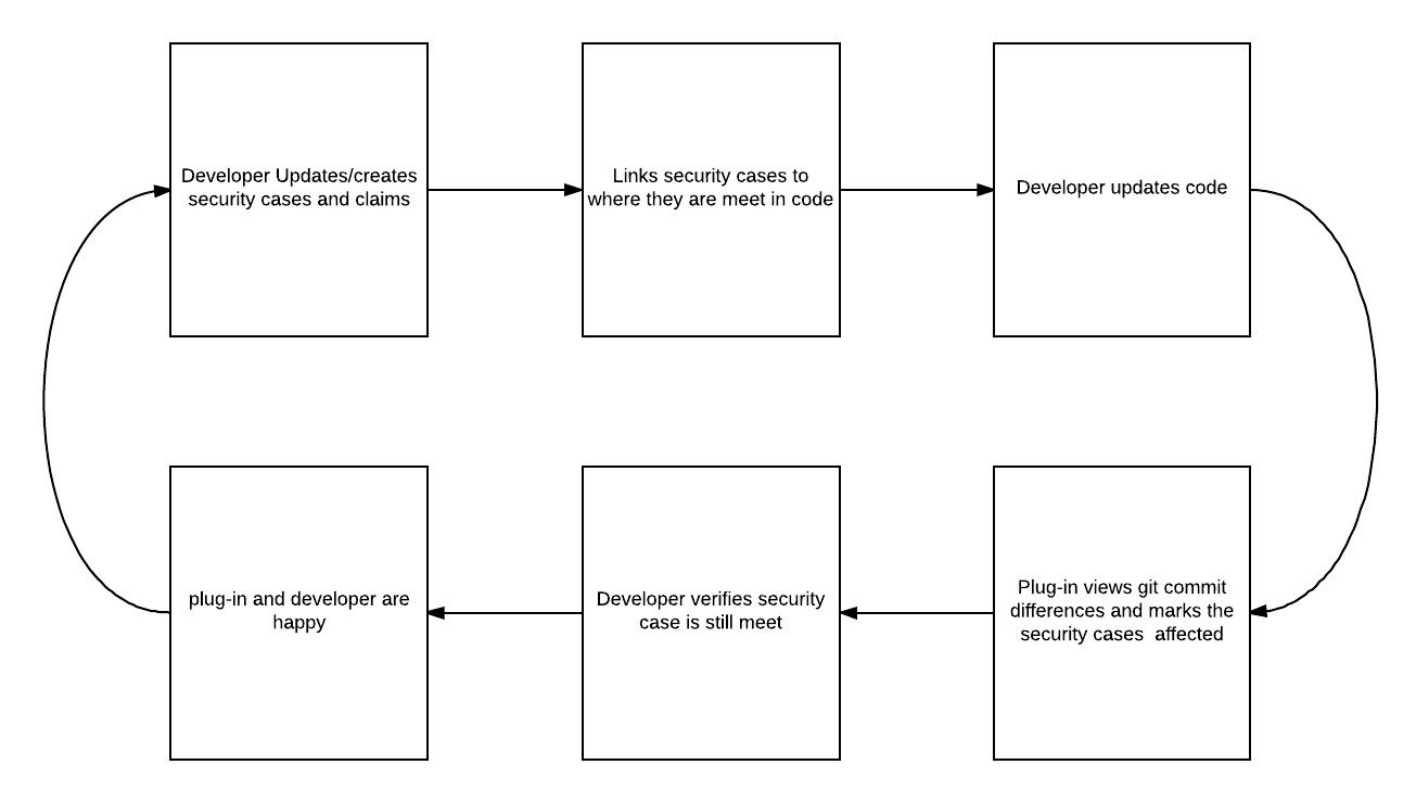
Presentation Outline

1. General Project Information
    a. Problem Statement
    b. General Concept
2. Requirements
    a. Functional
    b. Nonfunctional
    c. Technology
3. Design
    a. Development Timeline
    b. Implementation
4. Testing
5. Project Results and Use Samples

Problem

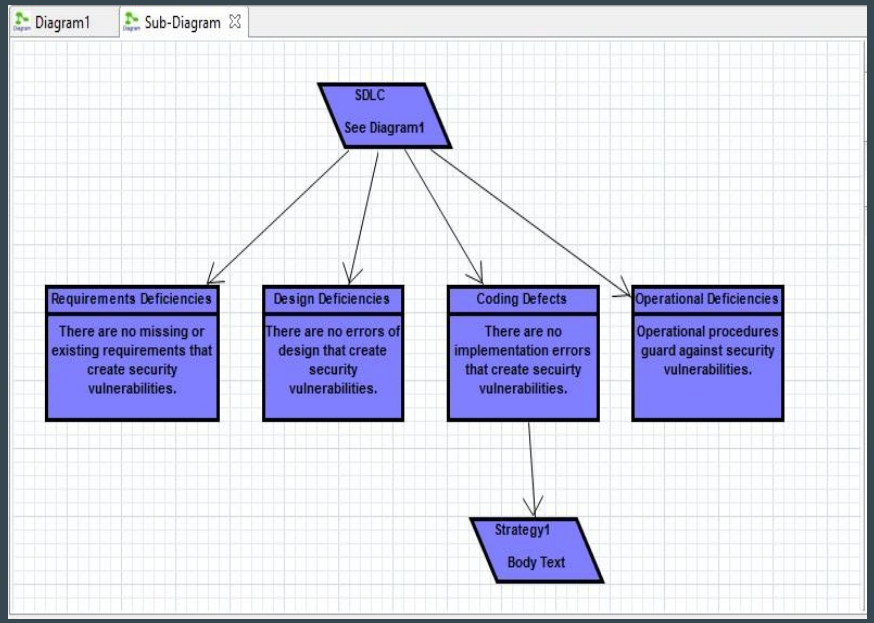
- Ensuring continued security can take significant time

General Concept



Project

- Develop eclipse plugin
  - design and manage security assurance cases.
- Enable users to
  - create, edit, and save/load assurance case diagrams
  - Nodes represent specific claims or evidence
  - Nodes affected by changes marked visually.

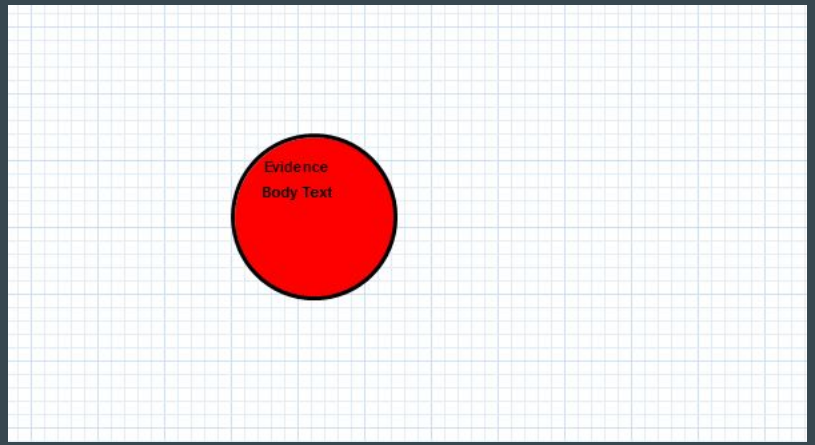


Functional Requirements

1. Add/Remove graphical elements to assurance case diagram
2. Multiple graphical elements
    - Case Claim
    - Evidence
    - Edges
3. User Interaction with elements
    - Move elements
    - Edit Text
    - Connect elements

Functional Requirements - Cont.

4. Connect graphical elements to function entry points
5. Visually alert user when claim and test are potentially no longer valid
6. Save/Load diagrams.



Non functional requirements

1. Visually Pleasing and Professional Interface
    - This software, while a school project, is hypothetically useful to companies that expect a certain level of professionalism.
2. Simple, easy to use
    - This software is intended to replace and improve on existing processes, so it must be better, easier to use, and faster than existing products.
3. Secure and bug free
    - Security assurance cases represent the security of other software, and so compromises or bugs in our program can expose risks in unrelated code.

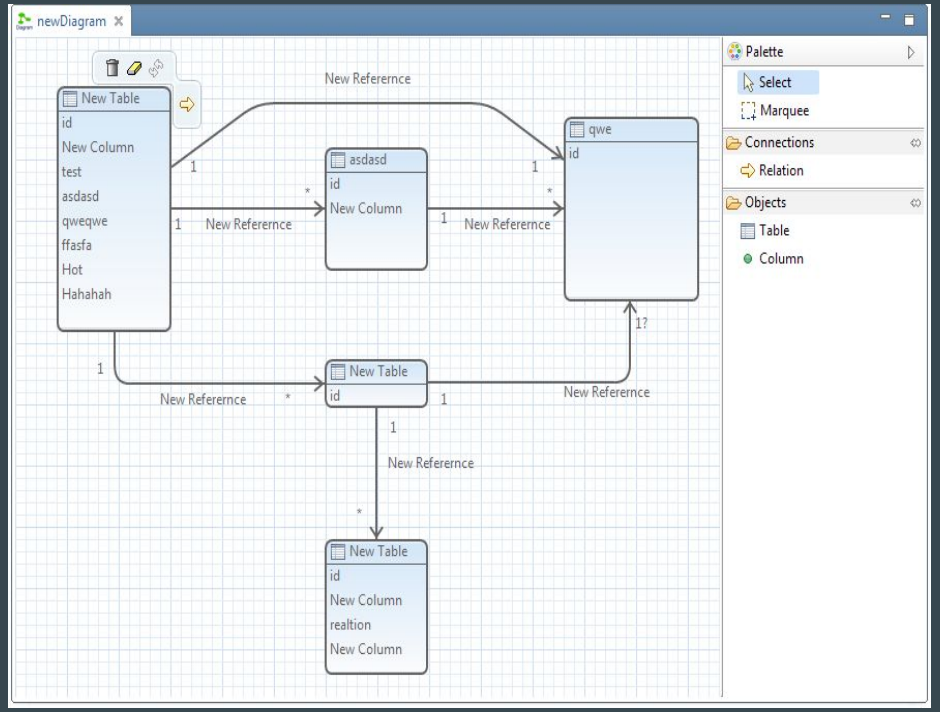
Technology Selection and Requirements: Groundwork

- Framework for entire project
- Capable of handling all project requirements
- Long researched decision

Name	Pros	Cons	Evaluation
JGraphx	Would be able to meet the requirements of the project. Would be easy to begin development with.	Built on Java Swing. Not eclipse specific. Dated aesthetic.	We decided that while this framework was a candidate as it met our requirements, it was not the best option available.
Eclipse Graphiti	Built for making eclipse plug ins. Aesthetically the most pleasing. Acceptable documentation and tutorials.	Team lacks experience with plug in development.	The team has chosen Graphiti for development of the project.
JGrapht	Would've been likely able to meet requirements in most ways.	Built on Java Swing. Not intended for the exact style of diagram necessary.	This framework was not considered a candidate compared to the previous options.
Graphviz	Able to draw diagrams needed.	Poor available information. Doesn't appear to be built for Java or Eclipse.	As with JGrapht, this library was not considered a candidate.

Technology Selection and Requirements: Graphiti (Cont.)

- What?
  - Designed to build diagram/chart based applications
- Why?
  - Meant for chart based applications!
  - Versatile and feature rich
- Why Graphiti?
  - Graphiti is designed for Eclipse plugins
  - Most professional looking of the options



Technology Selection and Requirements: WALA

- What?
  - Static analysis tool
  - Useful for generating call graphs
  - Officially supports Java and Javascript
- Why?
  - Helpful in validating claims
    - Can determine method dependencies for evidence supporting claims
    - If the call graph changes, may invalidate claim and flag in the system
- Why WALA?
  - The only library that had the functionality we needed

Technology Selection and Requirements: JGit

- What?
  - Java implementation of Git
- Why?
  - Allows us to analyze when methods change
- Why JGit?
  - Few to no alternatives to provide similar functionality
  - Easily implemented into our Java based project to execute git commands

Design - Development Timeline

- The first semester was dedicated to;
  - Gaining some understanding of the project domains (security assurance cases, Eclipse plugin development)
  - Implementing the diagram editor and fulfilling UI centered requirements
- This semester has been focused on;
  - Fixing some minor outstanding bugs on the diagram editor
  - The Git/Wala verification feature

Design - Implementation

Graphiti

- Centers around a feature provider
- Using feature provider, we implemented custom features
- Feature provider also let us override existing features to meet project needs

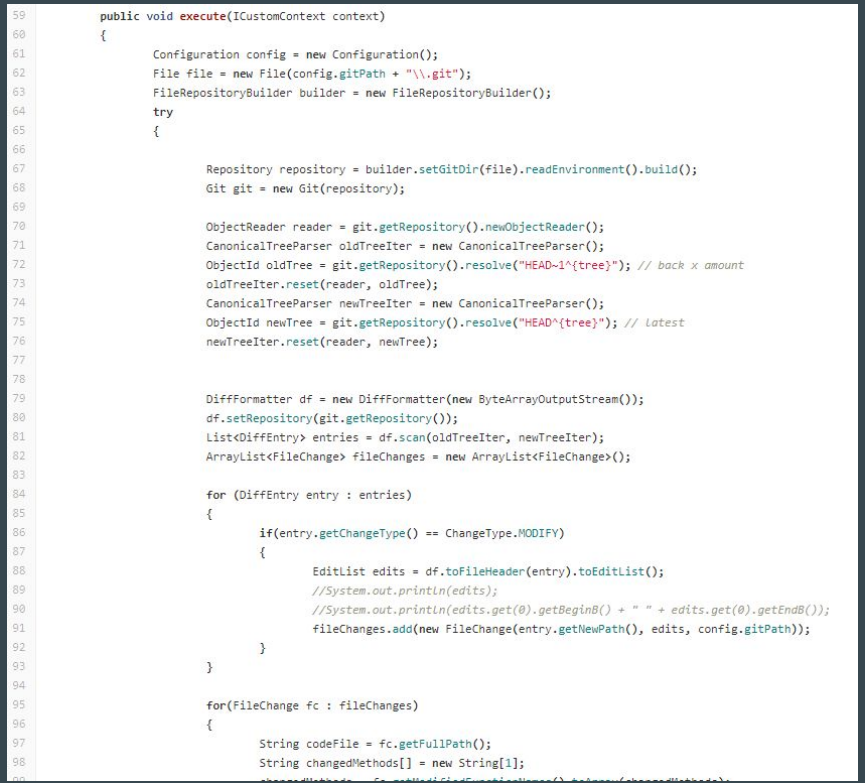
SeniorDesign | SeniorDesignFeatureProvider.java

```
73  @Override
74  public IAddFeature getAddFeature(IAddContext context)
75  {
76
77      if (context instanceof IAddConnectionContext)
78      {
79          return new AddConnectionFeature(this);
80
81      }
82
83      else if (context.getNewObject() instanceof EClass)
84      {
85          EClass eClass = (EClass) context.getNewObject();
86          if (eClass.getInstanceClassName().equals("Parallelogram"))
87              return new AddParallelogramEClass(this);
88          else if (eClass.getInstanceClassName().equals("Rectangle"))
89              return new AddRectangleEClass(this);
90          else if (eClass.getInstanceClassName().equals("Circle"))
91              return new AddCircleEClass(this);
92          else if (eClass.getInstanceClassName().equals("Oval"))
93              return new AddOvalEClass(this);
94      }
95
96      return super.getAddFeature(context);
97  }
98
99  @Override
100 public ICustomFeature[] getCustomFeatures(ICustomContext context)
101 {
102     return new ICustomFeature[]
103     { new RenameCustomFeature(this), new DrillDownCustomFeature(this), new AssociateDiagramCustomFeature(this) };
104 }
```

Design - Implementation

JGit

- Allows connection to Git repository through a config file
- Used Git diff between commits to get files changed and lines numbers
- Created our own Java and Javascript language parser to get methods changed based on Git diff information.



```
59      public void execute(ICustomContext context)
60      {
61              Configuration config = new Configuration();
62              File file = new File(config.gitPath + "\\.git");
63              FileRepositoryBuilder builder = new FileRepositoryBuilder();
64              try
65              {
66
67                      Repository repository = builder.setGitDir(file).readEnvironment().build();
68                      Git git = new Git(repository);
69
70                      ObjectReader reader = git.getRepository().newObjectReader();
71                      CanonicalTreeParser oldTreeIter = new CanonicalTreeParser();
72                      ObjectId oldTree = git.getRepository().resolve("HEAD~1^{tree}"); // back x amount
73                      oldTreeIter.reset(reader, oldTree);
74                      CanonicalTreeParser newTreeIter = new CanonicalTreeParser();
75                      ObjectId newTree = git.getRepository().resolve("HEAD^{tree}"); // latest
76                      newTreeIter.reset(reader, newTree);
77
78
79                      DiffFormatter df = new DiffFormatter(new ByteArrayOutputStream());
80                      df.setRepository(git.getRepository());
81                      List<DiffEntry> entries = df.scan(oldTreeIter, newTreeIter);
82                      ArrayList<FileChange> fileChanges = new ArrayList<FileChange>();
83
84                      for (DiffEntry entry : entries)
85                      {
86                              if(entry.getChangeType() == ChangeType.MODIFY)
87                              {
88                                      EditList edits = df.toFileHeader(entry).toEditList();
89                                      //System.out.println(edits);
90                                      //System.out.println(edits.get(0).getBeginB() + " " + edits.get(0).getEndB());
91                                      fileChanges.add(new FileChange(entry.getNewPath(), edits, config.gitPath));
92                              }
93                      }
94
95                      for(FileChange fc : fileChanges)
96                      {
97                              String codeFile = fc.getFullPath();
98                              String changedMethods[] = new String[1];
```

Design - Implementation

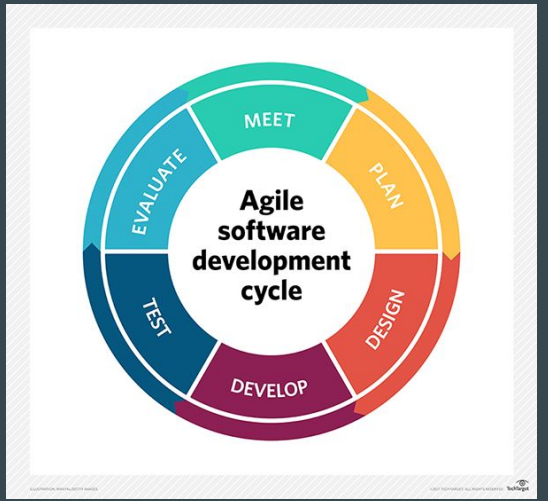
Wala

- Framework supplies classes to generate and use the call graph.
- Supports Java (source/binary) and Javascript (standalone/html embedded).
  - Dr. Othmane specified Javascript as the main focus.
  - HTML embedded was found to be the most stable implementation.

```
public class WalaUtil
{

	public static boolean isClaimValid(String function, String[] changedFunctions, String file)
	{

		JSCallGraphUtil.setTranslatorFactory(new CAstRhinoTranslatorFactory());

		CallGraph cg = null;

		try
		{
			cg = JSCallGraphBuilderUtil.makeHTMLCG(new File(file).toURI().toURL());
		} catch (IllegalArgumentException | IOException | CancelException | WalaException e)
		{
			e.printStackTrace();
		}

		if (cg != null)
		{
			for (int i = 0; i < changedFunctions.length; i++)
			{
				if (changedFunctions[i].equals(function))
				{
					return false;
				}

				for (CGNode node : cg)
				{

					if (node.getMethod().getSignature().contains(changedFunctions[i]))
					{

						Iterator<CGNode> temp = cg.getPredNodes(node);
						while (temp.hasNext())
						{
							CGNode cur = temp.next();
							if (cur.getMethod().getSignature().contains(function))
							{

								return false;
							}

						}
					}
				}
			}
		} else
		{
			System.out.println("Callgraph was null; unable to verify claims");
		}
		return true;
	}
}
```

Testing

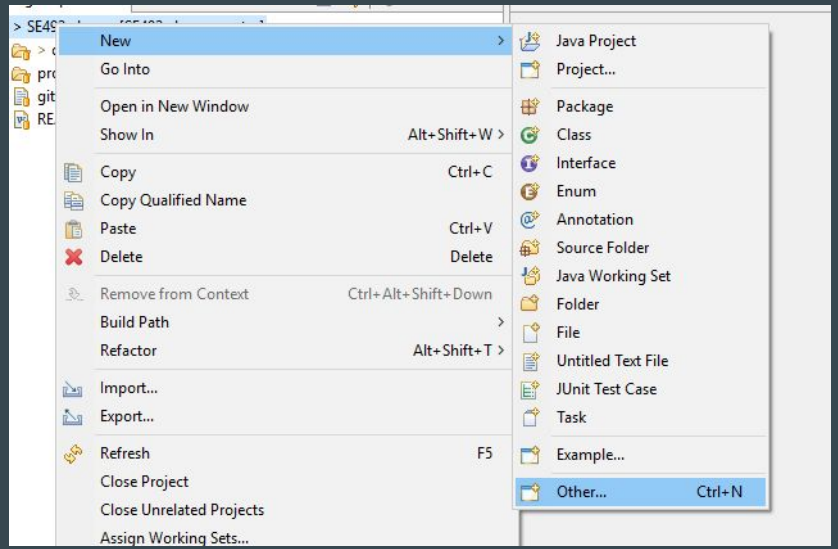
- Created bi-weekly sprints with Dr. Othmane
- Weekly, reported progress and get input
- Post sprint, we gave a demo to Dr. Othmane
- Dr. Othmane provided feedback to ensure product follows his vision/requirements

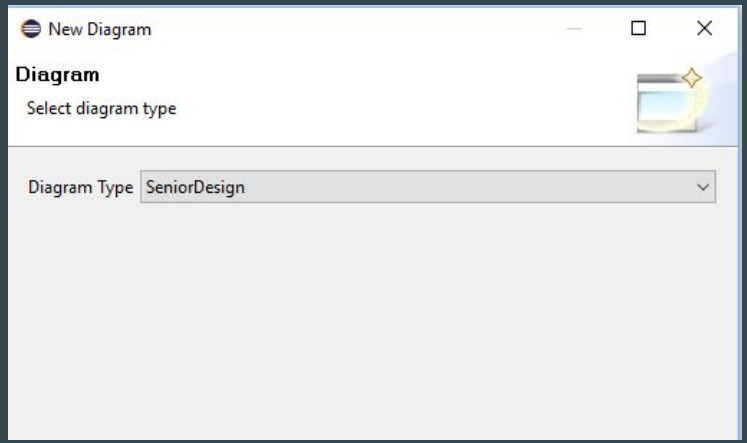


Current Status and Analysis

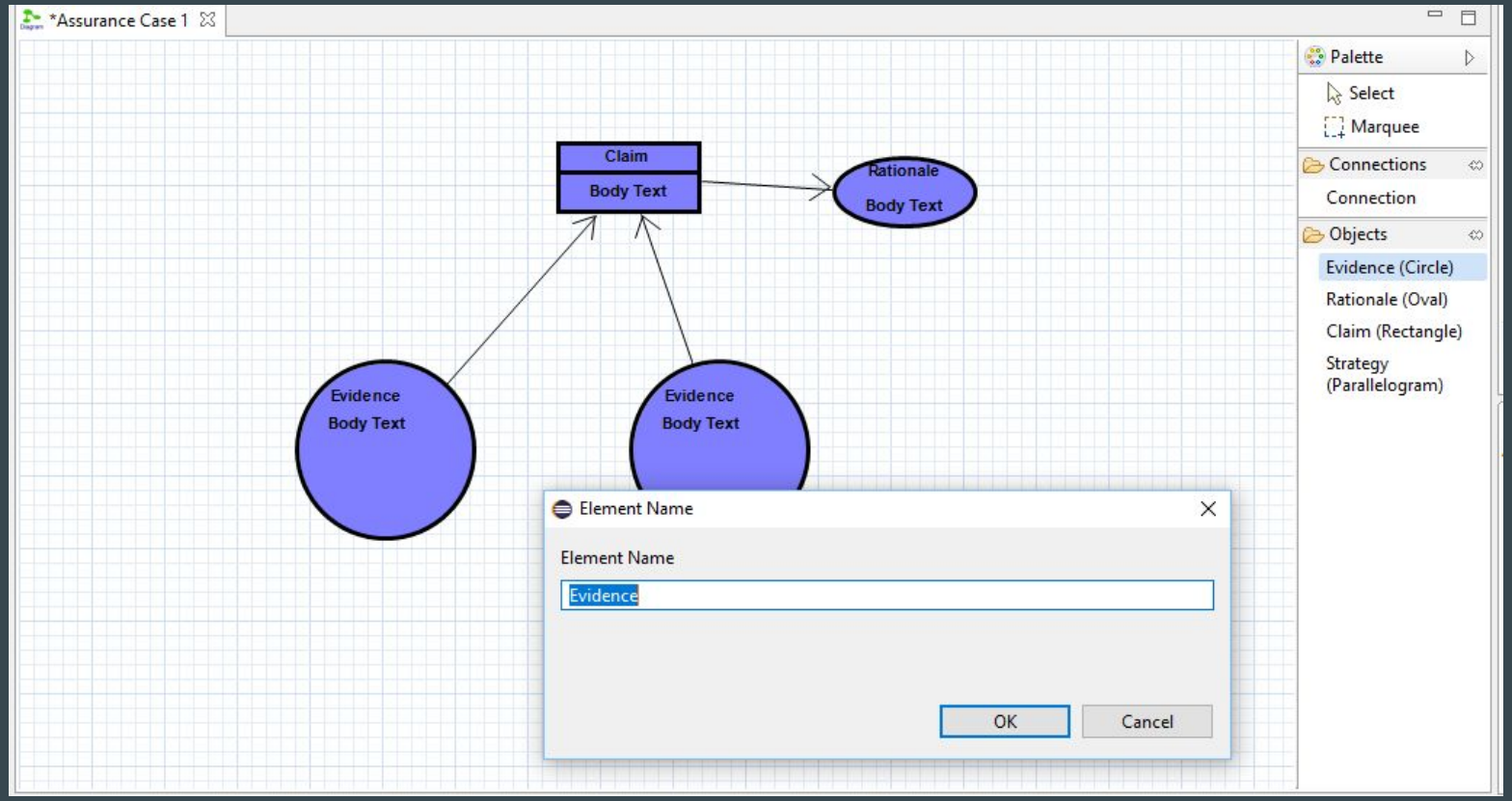
- Works for javascript present in html files
- Can be expandable to java and javascript files
- Pleased with current state

Project Demo - Create new diagram

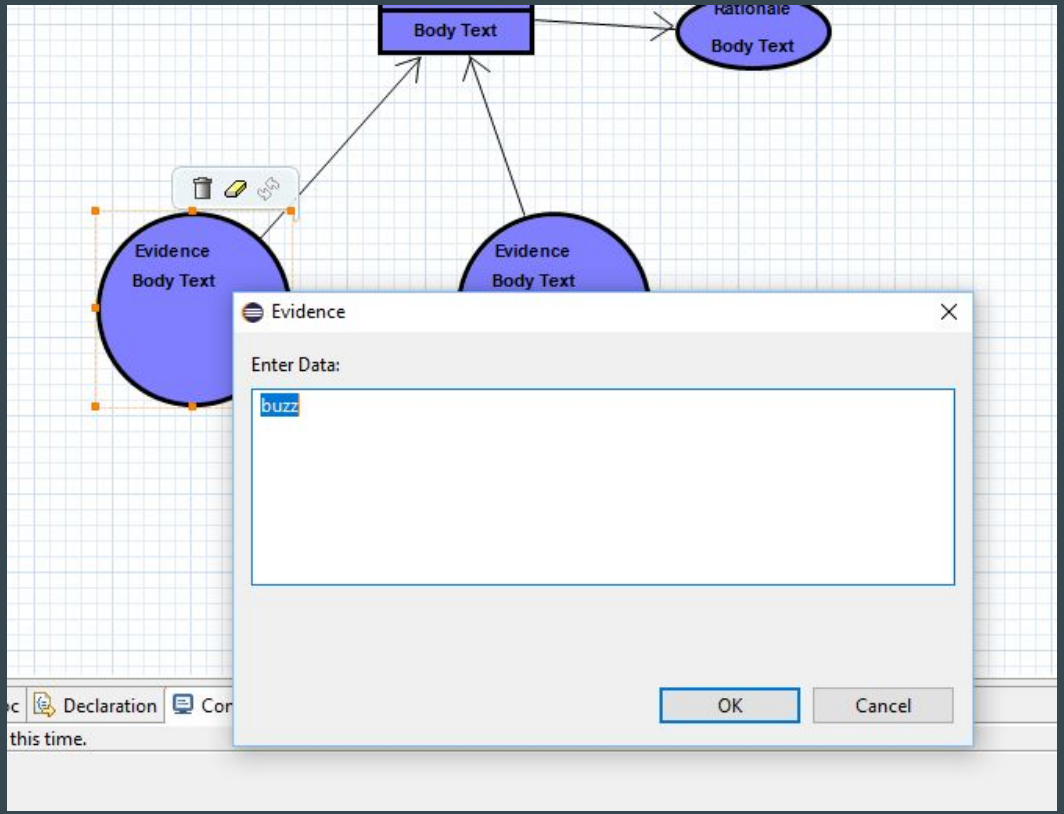




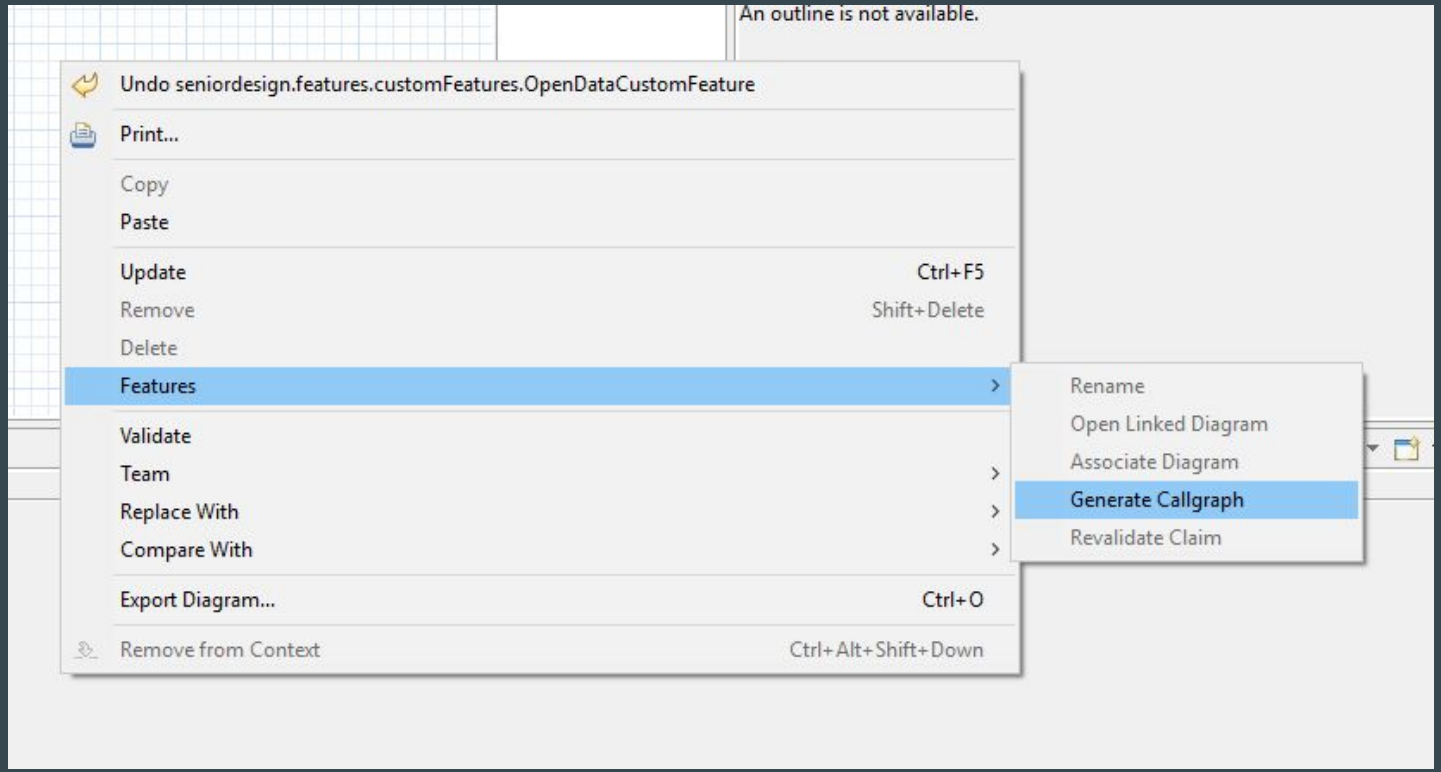
Make the Assurance Case



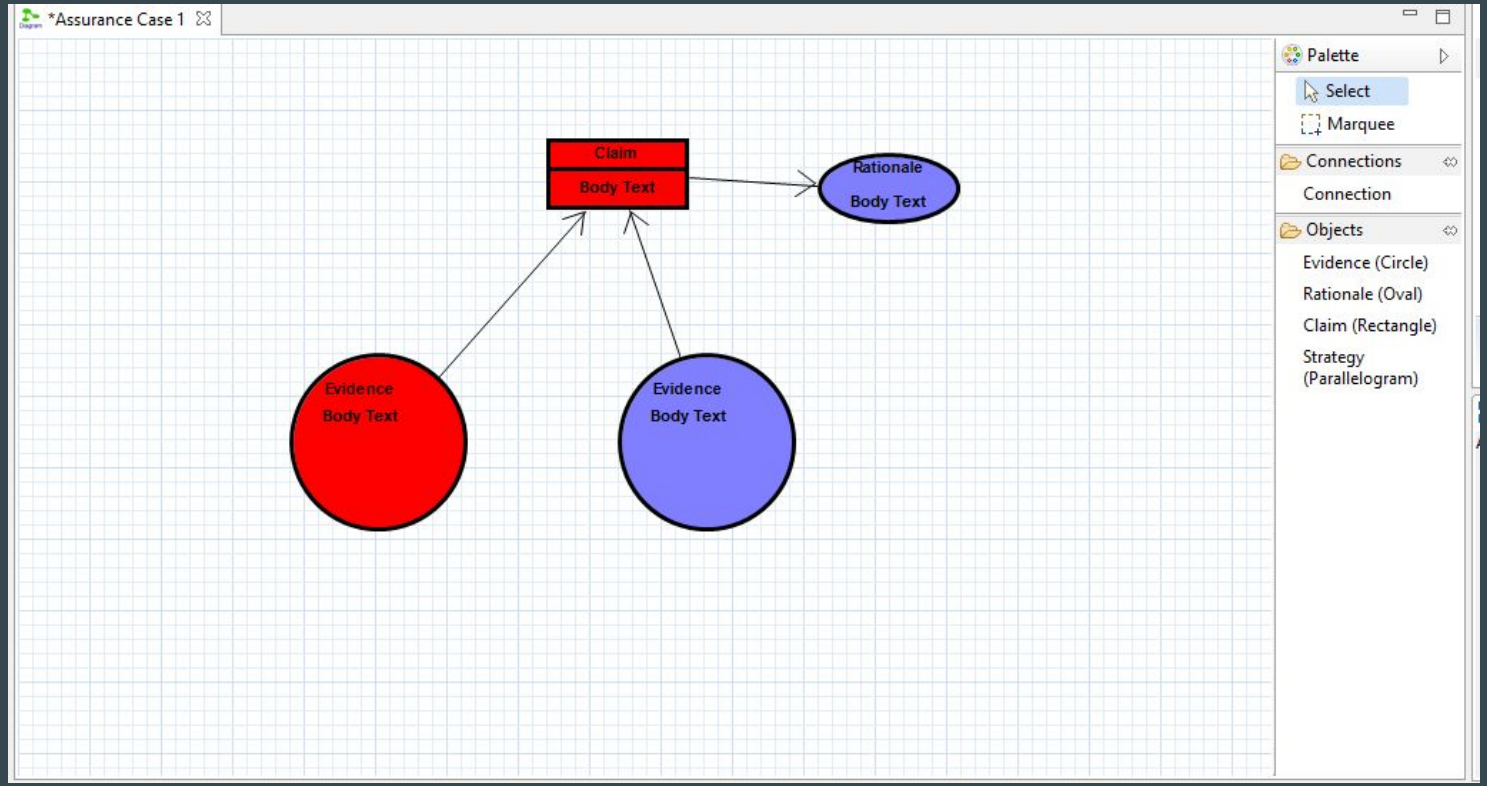
Add Entry Points



Generate Callgraph



Output


*Assurance Case 1
Claim
Body Text
Rationale
Body Text
Evidence
Body Text
Evidence
Body Text
Palette
Select
Marquee
Connections
Connection
Objects
Evidence (Circle)
Rationale (Oval)
Claim (Rectangle)
Strategy
(Parallelogram)