## CSC40232: SOFTWARE ENGINEERING

**Professor: Jane Cleland-Huang**
Term Projects + Some background
sarec.nd.edu/courses/SE2017

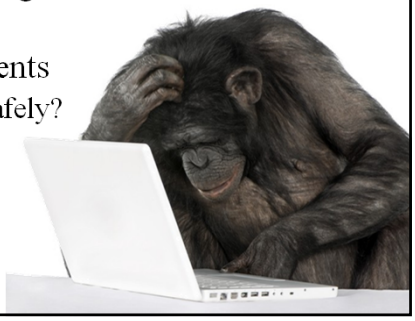
Department of
Computer Science and
Engineering

# Today's Goal

- Discuss the Projects we'll be working on and the domain of the projects.
  - I've posted all project descriptions to Sakai. These are draft documents.
- Discuss the environment (Eclipse).

## Program Comprehension

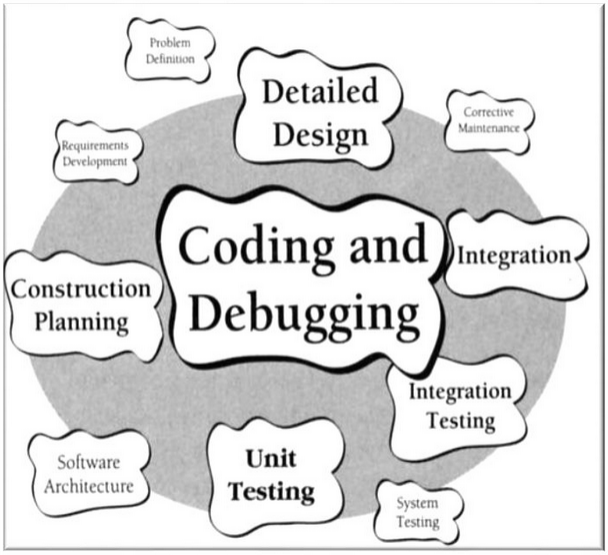
- Goes far beyond the ability to read syntax.
- Developers need to be able to truly understand source code in order to perform a diverse set of tasks:
  - Defect analysis and cause identification
  - Code inspection – understanding what the code actually does.
  - Code revisions and enhancements
    - Can a change request be made safely?
    - Understanding the original design intent (rationale) so that a change doesn't inadvertently impact the design quality.



## Supporting Developers

We can provide tools to support developers better understand the code.

- Design Rationales
- Context
- Trace Link Evolution
- Safety Assurance Case



# What is Traceability?

The ability to identify and document the lineage of each requirement, including its derivation (backward traceability), its allocation (forward traceability), and its relationship to other requirements.

*International Institute of Business Analysis*
*Body of Knowledge*
*Version 2.0*

Traceability is of particular concern in safety & mission-critical systems.

5

# Why Trace?

- Requirements validation
- Coverage analysis
- Change impact analysis
- Code comprehension
- Preservation of architectural knowledge
- Test regression selection
- Compliance verification
- Safety Analysis



6

## Standards Require it

The **Federal Aviation Administration**'s (FAA) DO-178B standard specifies that at each and every stage of development "software developers must be able to demonstrate traceability of designs against requirements.

The U.S. **Food and Drug Administration** (FDA) states that traceability analysis must be used to verify that the software design implements all of the specified software requirements, that all aspects of the design are traceable to software requirements, and that all code is linked to established specifications and test procedures.

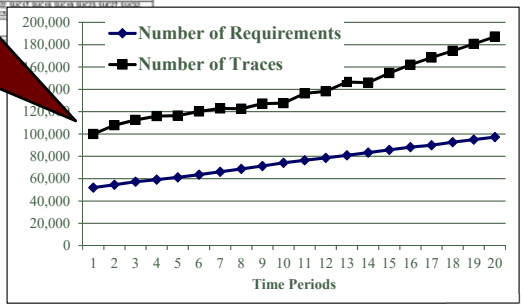
**Process improvement processes** such as CMMI also require traceability.

7

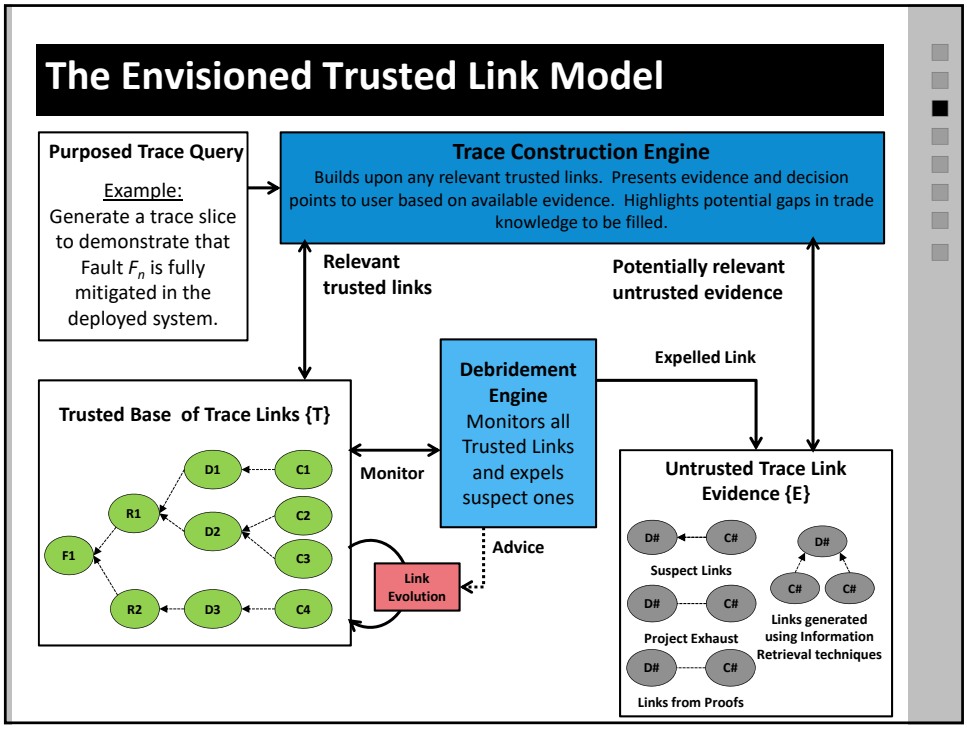
## ..but

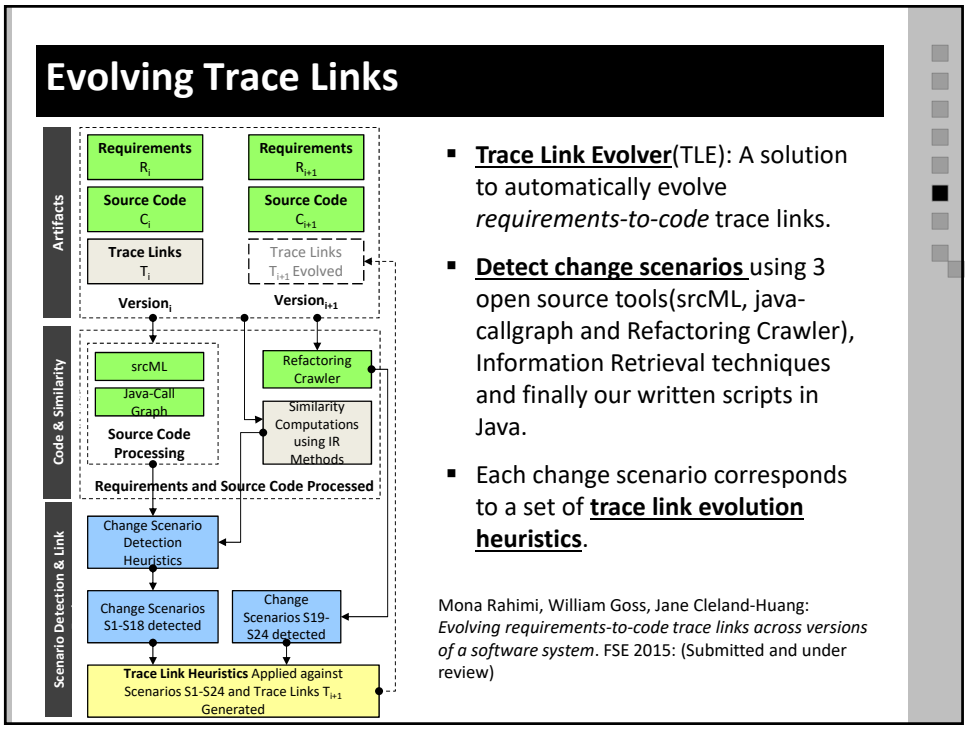

Excessive numbers of traceability links deteriorate into an unwieldy, inaccurate, tangle of relationships.

The number of requirements and other artifacts grow quickly as the project progresses.




8

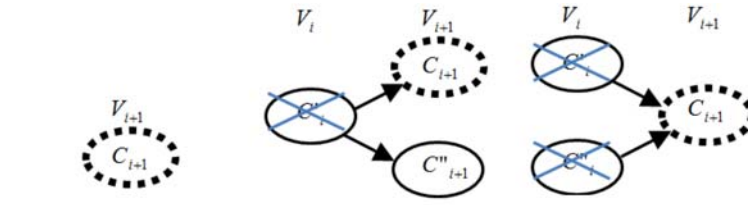



- **Trace Link Evolver**(TLE): A solution to automatically evolve *requirements-to-code* trace links.
- **Detect change scenarios** using 3 open source tools(srcML, java-callgraph and Refactoring Crawler), Information Retrieval techniques and finally our written scripts in Java.
- Each change scenario corresponds to a set of **trace link evolution heuristics**.

Mona Rahimi, William Goss, Jane Cleland-Huang: *Evolving requirements-to-code trace links across versions of a software system*. FSE 2015: (Submitted and under review)

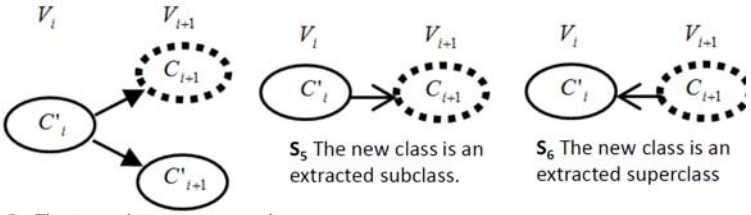
## Scenarios Related to "Added Class"




**$S_1$** The new class is a result of entirely new functionality.

**$S_2$** The new class was extracted from an existing one.

**$S_3$** The new class represents a merging of existing ones.




**$S_4$** The new class was created as a result of promoting a method.

**$S_5$** The new class is an extracted subclass.

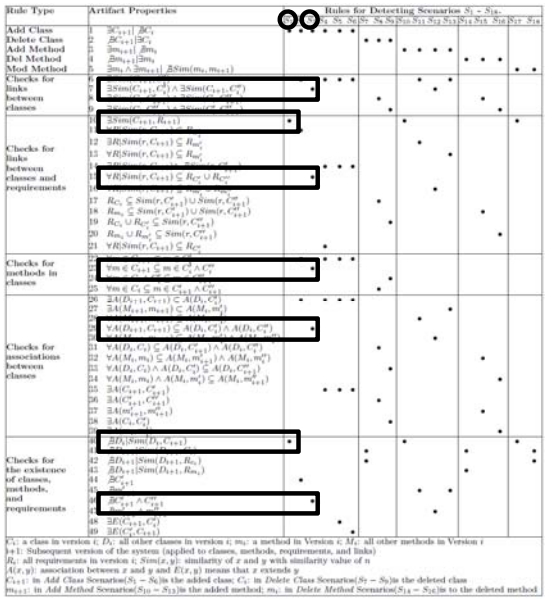
**$S_6$** The new class is an extracted superclass

## Covered Scenarios

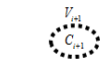
| Added Class | Deleted Class | Added Meth. |
|---|---|---|
| $S_1$ NewFunc. | $S_7$ ObsoleteFunc. | $S_{10}$ NewFunc. |
| $S_2$ ExtractedClass | $S_8$ DividedClass | $S_{11}$ ExtractedMeth. |
| $S_3$ MergedClasses | $S_9$ MergedClasses | $S_{12}$ MergedMethods |
| $S_4$ PromotedMethod | | $S_{13}$ DividedMethod |
| $S_5$ ExtractedSubclass | | |
| $S_6$ ExtractedSuperclass | | |

| Deleted Meth. | Changed Meth. | Basic |
|---|---|---|
| $S_{14}$ ObsoleteFunc. | $S_{17}$ NewFunc. | $S_{19}$ RenameClass |
| $S_{15}$ DividedMethod | $S_{18}$ ObsoleteFunc. | $S_{20}$ ChangeMethodSig |
| $S_{16}$ MergedMethods | | $S_{21}$ MoveMethod |
| | | $S_{22}$ PullupMethod |
| | | $S_{23}$ PushdownMethod |
| | | $S_{24}$ Ren. Method |

## We check properties to infer events

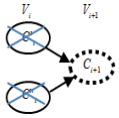


**Scenario 1:** New class added new functionality



Rule 10
Rule 40

**Scenario 3:** New class is the merging result of two old classes



Rule 7
Rule 15
Rule 23
Rule 29
Rule 46

## Link Heuristics evolve links..

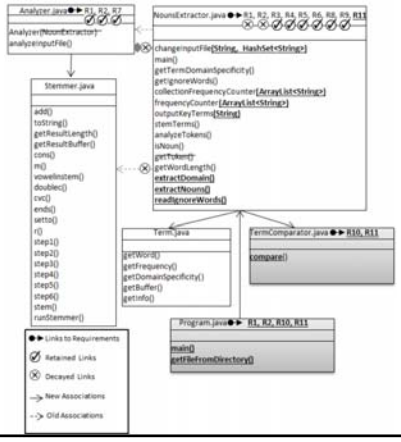
| | Link Evolution Actions |
|---|---|
| S1 | $l(C_{i+1}, R_{i+1}) \mid Sim(C_{i+1}, R_{i+1})$ |
| S2 | $l(C_{i+1}, R_{C'_i})$ |
| S3 | $l(C_{i+1}, R_{C'_i} \cup R_{C''_i})$ and $\nexists l(C'_i, R_{C'_i})$ and $\nexists l(C''_i, R_{C''_i})$ |
| S4 | $l(C_{i+1}, R_{m_i})$ and $\nexists l(C'_{i+1}, R_{m_i})$ |
| S5 | $l(C_{i+1}, R_{C'_i})$ |
| S6 | $l(C_{i+1}, R_{C_i})$ |
| S7 | $\nexists l(C_i, R_{C_i})$ |
| S8 | $l(C''_{i+1}, R_{C_i})$ and $l(C''_{i+1}, R_{C_i})$ and $\nexists l(C_i, R_{C_i})$ |
| S9 | $l(C''_{i+1}, R_{C_i})$ and $l(C''_{i+1}, R_{C'_i})$ and $\nexists l(C_i, R_{C_i})$ and $\nexists l(C'_i, R_{C'_i})$ |
| S10 | $l(m_{i+1}, R_{i+1}) \mid Sim(m_{i+1}, R_{i+1})$ |
| S11 | $l(m_{i+1}, R_{m'_i})$ |
| S12 | $l(m_{i+1}, R_{m'_i} \cup R_{m''_i})$ and $\nexists l(m'_{i+1}, R_{m'_i})$ and $\nexists l(m''_{i+1}, R_{m''_i})$ |
| S13 | $l(m_{i+1}, R_{m'_i})$ and $l(m''_{i+1}, R_{m'_i})$ and $\nexists l(m'_i, R_{m'_i})$ |
| S14 | $\nexists l(m_i, R_{m_i})$ |
| S15 | $l(m'_{i+1}, R_{m_i})$ and $l(m''_{i+1}, R_{m_i})$ and $\nexists l(m_i, R_{m_i})$ |
| S16 | $l(m''_{i+1}, R_{m_i})$ and $l(m''_{i+1}, R_{m'_i})$ and $\nexists l(m_i, R_{m_i})$ and $\nexists l(m'_i, R_{m'_i})$ |
| S17 | $l(m_{i+1}, R_{m_i}) \mid Sim(m_{i+1}, R_{m_i})$ |
| S18 | $\nexists l(m_i, R_{m_i}) \mid \nexists Sim(m_{i+1}, R_{m_i})$ |
| S19 | $l(C'_{i+1}, R_{C_i})$ and $\nexists l(C_{i+1}, R_{C_i})$ |
| S20 | *null* |
| S21 | $l(C'.m_i, R_{m_i})$ and $\nexists l(C.m_i, R_{m_i})$ |
| S22 | $l(C'.m_i, R_{m_i})$ and $\nexists l(C.m_i, R_{m_i})$ |
| S23 | $l(C'.m_i, R_{m_i})$ and $\nexists l(C.m_i, R_{m_i})$ |
| S24 | $l(m'_{i+1}, R_{m_i})$ and $\nexists l(m_{i+1}, R_{m_i})$ |

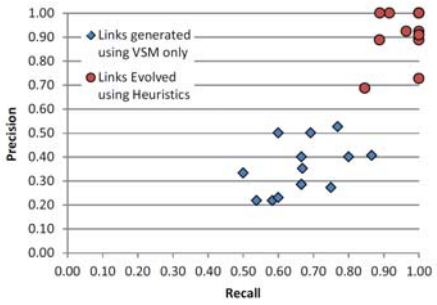
- **Scenario 1 is detected:** Create trace links between the related requirements and new class.

- **Scenario 3 is detected:** Create trace links to the requirements already linked to the two old merged classes and make the previous links to the two old classes obsolete.

## Results from an initial study

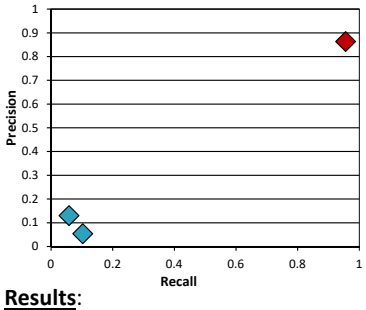
- 13 developers modified code and added new features.
- Answer trace matrix created.
- Results compared for generating trace links from scratch vs. evolving them.






## Results from Cassandra Study

- Evolved trace links across 27 versions of Cassandra.
- Starting Features: 48, $V_{Start}$: 488 Classes, $V_{Final}$: 702 classes
- 3 major revisions, 23 minor ones. Average # changes: 23.5




**Results**:

**TLE**: Recall: 0.956, Precision: 0.863

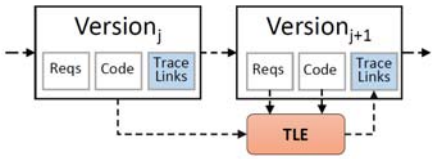
**VSM**: Recall: 0.103, Precision: 0.054

**LSI**: Recall: 0.058, Precision: 0.130

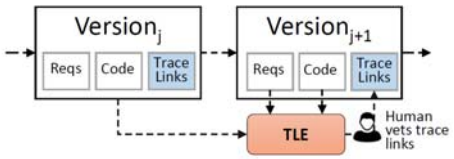
F1: An authenticator handles the authentication challenge/response cycles of a single connection.
**Classes**:
AllowAllAuthenticator
Permission etc.

F3: Cassandra provides a built-in row cache for super-fast access to frequently requested data, competitive with standalone caching products.
**Classes**:
FreeableMemory,
ConcurrentLinked
HashCache etc

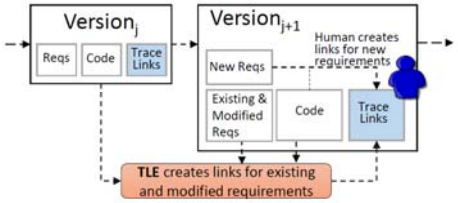
## Results from Dronology Study



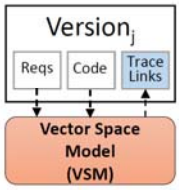

(a) TLE used to evolve links with no human intervention at any intermediate stage. Link errors are propagated.




(b) Links evolved using TLE are vetted by a human. The baseline for evolving links between each pair of versions is therefore assumed to be correct.

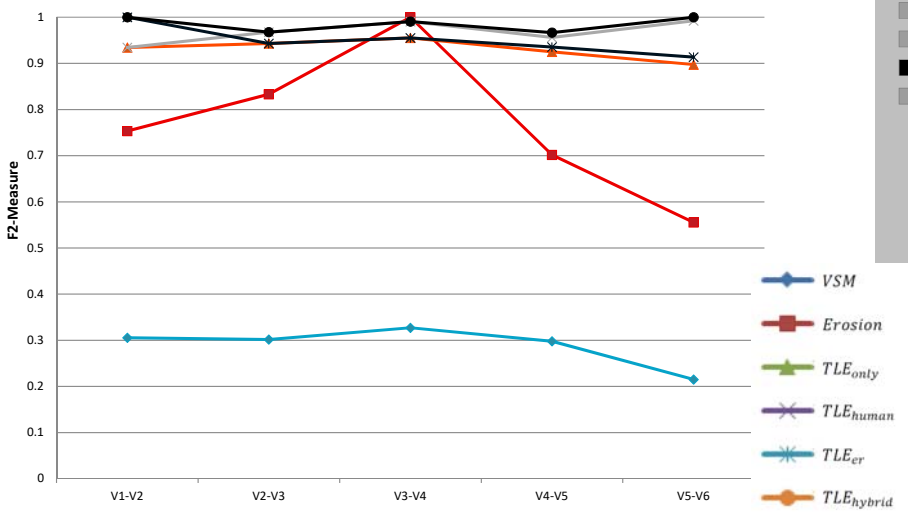



(c) A human creates links manually (or using an automated approach + manual vetting) for all new requirements.




(d) Baseline: Trace links are created from scratch for each version using VSM

## Results from Dronology Study



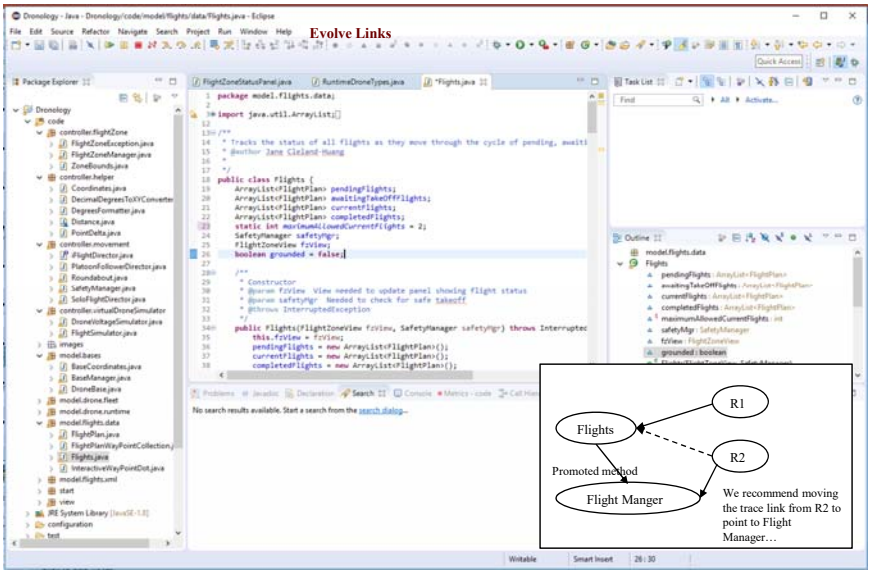
# Why Trace?

- Requirements validation
- Coverage analysis
- Change impact analysis
- **Code comprehension**
- Preservation of architectural knowledge
- Test regression selection
- Compliance verification
- **Safety Analysis**

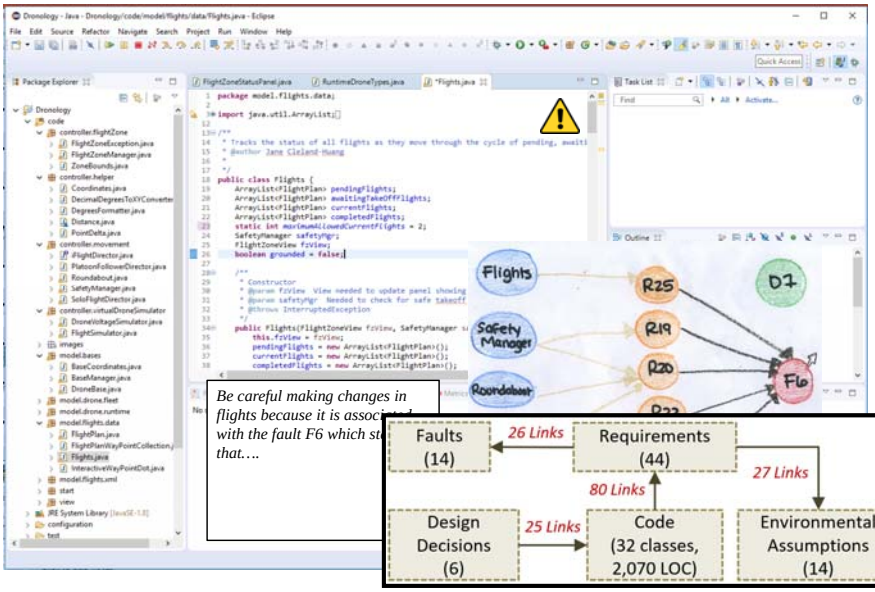


19

# Evolving Links



# Context Support for Maintenance




# Design Rationales

- Software lifecycle is very long and maintenance costs are very high.
- Original designers are unlikely to be available.
- Design Rationales help maintainers find problems, fix problems, and extend software with less risk
  - Captures designer's intent !
  - Avoids duplicating past effort by providing alternatives already considered !
  - Avoids repeating past mistakes by documenting when something was tried and failed
  - Helps developer understand how the code works.

# Design Rationales

A **design rationale** is an explicit documentation of the reasons behind decisions made when designing a system or artifact. As initially developed by W.R. Kunz and Horst Rittel, design rationale seeks to provide argumentation-based structure to the political, collaborative process of addressing wicked problems.
Wikipedia!

In its basic form, design rationales could allow us to describe a set of design decisions and their rationales and trade-offs.

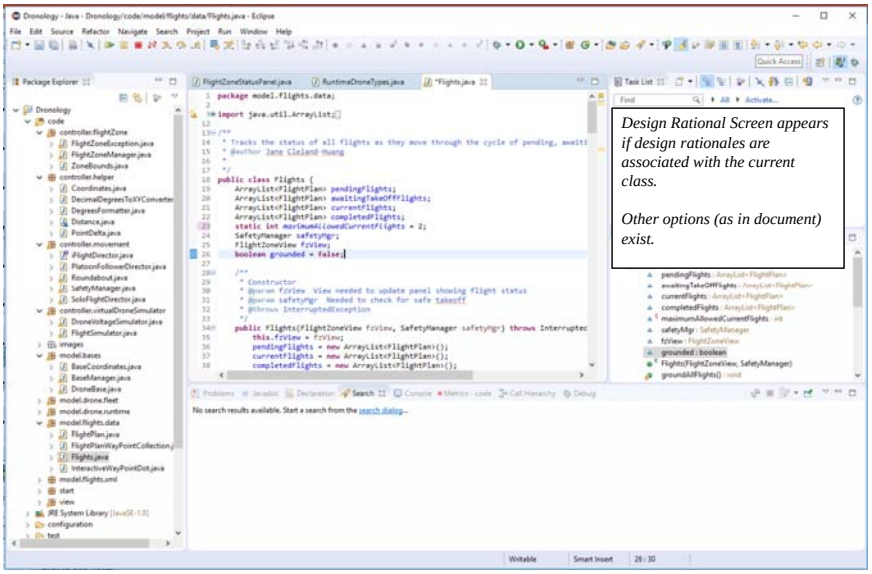
Formal argumentation structures are possible – but likely won't be used in practice.

*What are the design decisions associated with this code class?*

*Reasons that the decisions were made.*

*Alternate design decisions that were considered (if any) and their associated tradeoffs.*

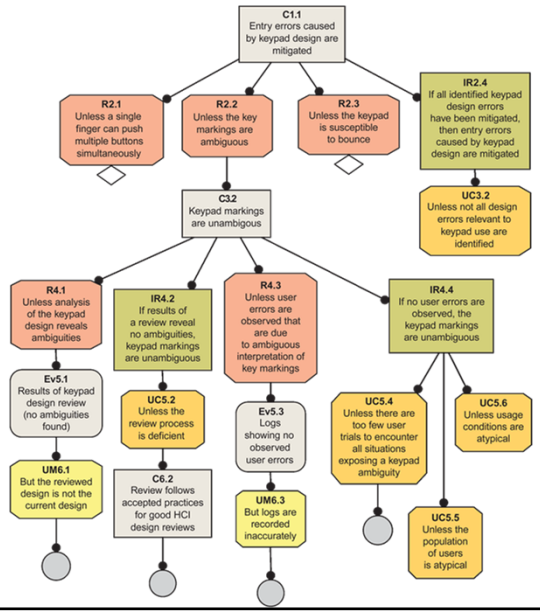
# Context Support for Maintenance



*Design Rational Screen appears if design rationales are associated with the current class.*

*Other options (as in document) exist.*

# Safety Assurance Cases




This project is different from the others. Instead of building an Eclipse plug-in you'll use the Eclipse framework to build an application to construct Safety Assurance Cases.