SEURAT: Software Engineering Using design RATionale

Janet Burge
Dissertation Defense

Advisor: David C. Brown
Committee: George Heineman, Carolina Ruiz, Feniosky Peña-Mora (UIUC)
What is Design Rationale?

“Design rationales include not only the reasons behind a design decision but also the justification for it, the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision”

[Lee, 1997]

The whole story behind the design, not just a static snapshot of the final product!
The Ham Story
Decision: How do we represent a conference room in the system?

Alternative: store the name (location) as a string
- Argument for: simple to code
- Argument against: difficult to extend

Alternative: create a conference room class
- Argument for: can contain information other than location
Why is DR valuable?

- 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
Why isn’t DR used now?

- Collection can impede design process
- Collection is often tedious
- Designers are reluctant to record “mistakes”
- Collection is very costly
- Not enough examples of use to provide motivation
Issues with Design Rationale

Capture

• Is there a non-intrusive way to capture rationale?
• How do you determine what to capture?
• Defines needs
• Provides incentive

Representation

• How should rationale be saved for later access?

Use

• Defines needs
• What are the uses of rationale?
• Which portions of rationale are the most useful?
Using Rationale to Support Software Development

- Focus on Software Maintenance:
  - Software lifecycle is very long
  - Maintenance costs are high
  - Original designers are unlikely to be available

- Rationale supports inference to help maintainers find problems, fix problems, and extend software with less risk
Our Hypothesis

- With appropriate tool support, rationale can provide useful support to the software maintainer.
  - Improved efficiency – less time required to perform maintenance tasks
  - Improved effectiveness – rationale assists maintainers in making better decisions
SEURAT: Software Engineering Using RATionale

- Using rationale to assist in software development and maintenance:
  - verifying consistency and completeness of the rationale
  - evaluating the support for design alternatives
  - ensuring that rejected alternatives are not repeated
  - presenting applicable rationale to the maintainer to assist in modification
  - maintaining rationale consistency by propagating results of rationale modifications
SEURAT Capabilities

- Tight integration with development/maintenance environment
- Allows “what if analysis” of
  - changing design priorities
  - disabling assumptions
  - disabling requirements
- Supports traceability of requirements to decisions (and then to code)
Key SEURAT Issues

- **Inference** – how can we inference over the rationale to support uses that go beyond presentation?
- **Representation** – what needs to be represented to support inference?
- **Ontology** – how do we provide a common vocabulary to support inferencing over content?
- **Integration** – how can we encourage rationale use by integrating with an existing development environment?
Inference

- Produce hypothesis, not conclusions
- Two categories:
  - Syntactic inference that uses the structure of the rationale
  - Semantic inference that uses the content
- Syntactic inference over structure:
  - Look for decisions with no selected alternative
  - Look for selected alternatives with no supporting arguments
  - Check for unanswered questions
Inference (cont.)

- Semantic inference over content:
  - Evaluate alternatives and alert if weaker alternatives are selected
  - Re-evaluate decisions after an assumption is disabled
  - Re-evaluate decisions if argument priorities change
  - Check for tradeoff violations
  - Check for dependencies between alternatives
  - Check for requirement violations
Rationale Representation

- Argumentation Representation
  - Semi-structured representation that is readable by machines and people
  - Captures the arguments for and against each alternative
  - Supports arguments about requirements, assumptions, claims (non-functional requirements), and other alternatives (dependencies)
Argument Ontology

- Provides a common vocabulary used to compare alternatives and arguments during inference
- List of common arguments for software changes at varying levels of abstraction
  - Based on the “ilities” (affordability, scalability, etc.)
- Each ontology entry has an associated default importance that can be inherited by rationale that refers to it
Argument Ontology

- Affordability Criteria
- Adaptability Criteria
- Dependability Criteria
- End User Criteria
- Usability
  - Increases Physical Ease of Use
    - {provides | supports} effective use of screen
    - minimizes keystrokes
    - {provides | supports} increased visual cues
    - is easy to read
  - Increases Cognitive Ease of Use
  - Increases Interface Consistency
  - Increases Recoverability
  - Increases Learnability
  - Increases Acceptability
  - Provides User Customization
  - Supports Internationalization
  - Increases Accessibility
- Integrity
- Needs Satisfaction Criteria
- Maintainability Criteria
- Performance Criteria
Integration

- SEURAT is implemented as an Eclipse plug-in
  - Rationale is more likely to be used if the developer does not need to switch tools
  - Rationale can be directly associated with code and its presence indicated in the editor used by the developer to write and maintain code

- Rationale is stored in a MySQL database
  - Scalable to large amounts of rationale
  - SQL queries support inference and presentation
Using SEURAT in Software Maintenance

- Reasons for Maintenance
  - Corrective maintenance (fixing errors)
  - Enhancive maintenance (new functionality)
  - Adaptive maintenance (non-functional enhancements)

- Sources of error
  - Requirement violations
  - Defects (bugs)
  - Changed or incorrect assumptions
  - ....
Rationale Explorer

Source Code Editor

Package Explorer

Rationale Task List
Select Assumption

calendar only shows four weeks
customer normally combines room and building
error messages are likely to be short
more text in box
new date class would not be much work
no way to know which room needed
not much more work than just checking password
prefer to minimize meeting retrievals
probably not very useful
read-only access might be useful
standard working hours 8 to 6
this will work

Search:
Select Assumption

calendar only shows four weeks
customer normally combines room and building

Assumption Information

Name: standard working hours 8 to 6
Description: Standard work day goes from 8 to 6
Enabled: [ ]

Save  Cancel
Select Assumption

calendar only shows four weeks

Assumption Information

Name: standard working hours 8 to 6

Description: Standard work day goes from 8 to 6

Enabled

Save  Cancel

Search

Show  Edit  Cancel
SEURAT Results

- Rationale Explorer
  - Indicates disabled assumption
  - Displays a warning for the decision that relies on the assumption
SEURAT Results

- **Rationale Task List**
  - Describes the warning
  - Allows the decision to be viewed in an editor
Finding the Implementation

- Alternative Editor
  - Specifies the class, method, or instance variable
Finding the Implementation

- **Bookmark List**
  - Lists the file, folder, and line number
  - Double-clicking brings up the code in the editor
Finding the Implementation

```java
private int numHalfHours = 20; // was 28
private int numDays = 5;
private int interval = 30;
private int START_DAY = Calendar.MONDAY;
private int END_DAY = Calendar.FRIDAY;
private int START_TIME = 8; // 8 am
private int MONTH_SPAN = 28;
private int WEEK_SPAN = 7;

private Frame frame;

public MeetingApp(String user) {
```
Evaluation

Three maintenance tasks:
- adaptive maintenance (a non-functional change)
- corrective maintenance (fixing a “bug”)
- enhancive maintenance (extending functionality)

Twenty subjects in two groups: experimental and control

Measures:
- time required to find the location in the code that needed changing and the time needed to complete the task (all subjects)
- usability survey (SEURAT group only)
- usefulness survey (SEURAT group only)
## Subject Distribution

<table>
<thead>
<tr>
<th></th>
<th>SEURAT</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Experience (Expert/Moderate/Some)</td>
<td>3/4/3 people, respectively</td>
<td>3/4/3 people, respectively</td>
</tr>
<tr>
<td>Average Work Experience</td>
<td>6.85 years</td>
<td>5.65 years</td>
</tr>
<tr>
<td>Eclipse Experience</td>
<td>60%</td>
<td>60%</td>
</tr>
</tbody>
</table>
Results (Time)

Average Time

Average Time (in Minutes)

Task

Adaptive Delta
Adaptive Total
Corrective Delta
Corrective Total
Enhance Delta
Enhance Total

Control
SEURAT
Results by Expertise

Enhancive Maintenance

Average Time to Complete (in Minutes)

Java Expertise

Control
SEURAT
Results (Usefulness Survey)

Usefulness Assessment

- Easier to Maintain Software
- Helped Find Associated Code
- Clear Error and Warnings
- Tasks Took Less Time
- Would Use SEURAT

Questions
Experiment Summary

- On average, SEURAT users outperformed the control group except for the “Expert” users.
- The learning curve was a factor.
- Variances in the control group were typically twice that in the SEURAT group.
- More experiments are needed!
Contributions

- Tightly integrated usable environment that supports rationale capture and use
- Argument ontology that contains common arguments for making software design decisions
- Rationale representation tailored to software engineering and maintenance
- Uses of rationale that go beyond presentation:
  - support for “what-if” inferencing
  - checking for rationale consistency and completeness
Related Work

- Includes:
  - Lee: Decision Representation Language
  - Peña-Mora: DRIM – Design Recommendation and Intent Model
  - Klein: C-Re-CS
  - Beñares-Alcántara, King: KBDS
  - Bose: Decision Ontology within the WinWin framework
  - Chung, et. al.: NFR-Framework
Summary and Conclusions

- Targeted software maintenance as an area that can best utilize rationale
- Demonstrated that with appropriate tool support, rationale can provide useful support to the software maintainer:
  - Demonstrated uses of DR that go beyond browsing and presentation
  - Integrated DR support with a standard software development environment
Future Work

- Expansion to additional design phases (such as associating rationale with design artifacts, e.g. UML diagrams)
- Enhanced support for rationale capture by integrating with other tools (such as Configuration Management systems)
- Study of multi-user rationale
- Additional experimentation with longer-term use, more subjects, larger projects
Acknowledgements

- My advisor: Dave Brown
- My committee: George Heineman, Carolina Ruiz, Feniosky Peña-Mora
- My experiment subjects
- My parents and my friends
- My co-workers at CRA
- My professors and colleagues at WPI
Questions and Discussion