Software Engineering Using Rationale

Janet E. Burge, Miami University
http://www.users.muohio.edu/burgeje/SEURAT/
What is Rationale?

• The rationale for a system describes the reasons behind decisions made during its development.
• Not just a snapshot of the final decision – the rationale contains all alternatives considered, arguments for and against them, and tradeoffs considered.
  – The argumentation behind the decision.
Example (from a Conference Room Scheduling System)

- **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 Rationale valuable?

- Captures the developer’s intent
- Avoids duplicating past effort by providing alternatives already considered
- Avoids repeating past mistakes by documenting when something was tried and failed
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)
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
Evaluation Using Rationale

- Look for decisions with no selected alternative
- Look for selected alternatives with no supporting arguments
- Check for unanswered questions
- 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)
Comparing Decision Alternatives

- Rationale Explorer shows alternatives for each decision.
- Selected alternatives are denoted by an “S”.
- Arguments are color coded: green is for, red is against, no color indicates rationale dependencies.
- Arguments can refer to requirements, claims (non-functional requirements), assumptions, and relationships to other alternatives.
Example: Analyzing Impact of Development Priorities

- Non-functional requirements (NFRs) may have a significant impact on the final software product.
- What if these change? How would that affect the software?
- The rationale captures the impact of NFRs on the software and can be used to determine what should be changed if the importance of an NFR, or NFRs, changes.
Analyzing Impact of Development Priorities

- Can analyze rationale to determine which NFRs were most frequently used in arguments
- Compare those criteria to user and developer goals – do they match?
- In this example, the most commonly used NFR was “Reduces Development Time”
package meeting;

import java.util.EventObject;

/**
 * This class represents a meeting event which either adds or removes a
 * from the schedule. It is used to tell the schedule to add or remove a
 * @author jburke
 *
 * int ADD = 1;
 * int CANCEL = 2;
 */

tObject extends EventObject {

// Class : MeetingEventObject

 // Development Cost
 // Uses Standard Tools and Environments
 // Uses Familiar Tools and Environments
 // Reduces Development Time
 // is component based
 // uses COTS/GOTS software
 // reduces customization
 // utilizes existing code developed
 // uses automatically generated code
 // Reduces Project Risk
 // Reduces Prototyping Cost
 // Reduces Risk Analysis Cost
 // Reduces Component Integration Cost
 // Reduces Domain Analysis Cost
 // Reduces Inspection Cost
 // Development Cost
 // Operating Cost
 // Maintenance Cost

 // Ontology Information

 Name: Reduces Development Time
 Description: Reduces the amount of time and effort to develop the software
 Importance: 10

 // Package Explorer

 // Problems Javadoc Declaration Bookmarks Rationale Task List

 // CRSCorrective
 // meeting
 // schedule
 // schedulingSystem
 // ConferenceSystem.java
 // CRSCorrective.java
 // EditUser.java
 // LoginUser.java
 // Message.java
 // ScheduleMeeting.java
 // UserInfo.java

 // Browser window

 // Rationale Explorer

 // Requirements
 // Decisions
 // Tradeoffs
 // Argument-Ontology
 // Affordability Criteria

 // Development Cost
 // Reduces Development Time
 // is component based
 // uses COTS/GOTS software
 // reduces customization
 // utilizes existing code developed
 // uses automatically generated code

 // Operating Cost
 // Maintenance Cost

 // Executable Code

 // int ADD = 1;
 // int CANCEL = 2;

 // // Class : MeetingEventObject

 // Reduces Development Time
 // Description: Reduces the amount of time and effort to develop the software
 // Importance: 10

 // CRSCorrective
 // meeting
 // schedule
 // schedulingSystem
 // ConferenceSystem.java
 // CRSCorrective.java
 // EditUser.java
 // LoginUser.java
 // Message.java
 // ScheduleMeeting.java
 // UserInfo.java

 // Browser window

 // Rationale Explorer

 // Requirements
 // Decisions
 // Tradeoffs
 // Argument-Ontology
 // Affordability Criteria

 // Development Cost
 // Reduces Development Time
 // is component based
 // uses COTS/GOTS software
 // reduces customization
 // utilizes existing code developed
 // uses automatically generated code

 // Operating Cost
 // Maintenance Cost

 // Executable Code

 // int ADD = 1;
 // int CANCEL = 2;
Analyzing Impact of Development Priorities

- Initially, there was only one warning that the best alternative was not selected.
- Two more warnings appear if “Reduces Development Time” becomes unimportant.
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
/**
 * Loads all meetings occurring between the start and end MeetingDate. This does not
 * generate a meeting event for each retrieved meeting.
 *
 * @param start
 * @param end
 */

public void loadMeetings(MeetingDate start, MeetingDate end) {
    Enumeration meetingEnum;
    MeetingObj newMeeting;

    meetingEnum = retrieveMeetings(start, end);

    while (meetingEnum.hasMoreElements()) {
        newMeeting = (MeetingObj) meetingEnum.nextElement();
        // create and send meeting event
        MeetingEventObject meo = new MeetingEventObject (this,
                                                  newMeeting.getStart(), newMeeting.getEnd(),
                                                  newMeeting.getInfo(), newMeeting.getOwner(), MeetingEventObj;
        fireAction(meo);
    }
}
```
SEURAT Summary

• Demonstrated that with appropriate tool support, rationale can provide useful support to the software developer:
  – Demonstrated uses of rationale that go beyond browsing and presentation
  – Integrated rationale support with a standard software development environment