

A Pattern for Modeling Rework in Software Development Processes

Aaron G. Cass

Department of Computer Science
Union College
Schenectady, NY

Leon J. Osterweil

Alexander Wise

Department of Computer Science
University of Massachusetts Amherst
Amherst, MA

Modeling Real Processes

- Goals:
 - Support reasoning
 - Support (partial) automation
- But, in real processes, rework is prevalent
- “Going back” does not capture what really happens

Motivating Examples

- Requirements Analysis
- Dispute Resolution
- Software Design

Requirements Analysis

- Scenario:
 - Developing a set of requirements elements
 - Name requirements element
 - Elaborate requirements element
 - Review requirements element
- Rework triggered by requirements review:
 - Not consistent with existing requirements
- Some requirements elements must be reworked

Requirements Analysis : Our Solution

- Re-invoke 'elaborate requirements element'
- Elaborate requirements element takes requirement to work on as parameter
 - Re-invocation(s) pass values that restrict/control
- When re-invocation complete, continue as before

Dispute Resolution

- Scenario:
 - Labor and Management in negotiation over a grievance
 - Brainstorm:
 - Each party offers options to resolve dispute
 - Each party labels options as acceptable or not
 - Determine set of mutually-acceptable options
- Rework if no mutually-acceptable options

Dispute Resolution (2)

- Two ways to carry forward:
 - Brainstorming further options
 - Each party offers options to resolve dispute
 - Each party labels options as acceptable or not
 - Determine set of mutually-acceptable options
 - Reconsider problem
 - Reframe the problem
 - Brainstorm options for the new problem

Dispute Resolution : Our Solution

- When rework is triggered, evaluate the situation
 - If impasse is reached, try reconsidering the problem
 - Otherwise, brainstorm some more
 - Both approaches re-invoke Brainstorm
- Brainstorm takes problem and negotiation history as parameters
 - Reconsider the problem passes new problem parameter
 - Brainstorm more passes same problem
 - Both pass negotiation history
- When re-invocation complete, continue as before

Software Design

- Scenario:
 - Developing a set of design elements
 - Name a design element
 - Define a design element
 - Review a design element
- Rework triggered if design inconsistent
 - Inconsistent with requirements elements
 - Inconsistent with existing design elements

Software Design (2)

- 2 options if design inconsistent wrt requirements
 - Rework design element
 - Rework requirements element
- “Ripple effect” possible
 - Re-elaborate requirements element
 - Review requirements element
 - Find inconsistencies with other design elements
 - Causing more rework of design elements or requirements elements

Software Design : Our Solution

- Design review triggers rework
- Then evaluate situation:
 - Decide to rework design or requirements
- Re-invoked tasks take parameters:
 - Elaborate requirements element
 - Define design element

Software Design : Our Solution (2)

- If we re-invoke elaborate requirements element
 - When done, review requirements element
 - Review can trigger further rework
 - In new context, trigger causes new response:
 - Evaluation of new situation
 - Decide to change requirements or design
- When ripple effect complete, continue as before

Recognizing a Pattern in the Examples

- Rework is triggered by a review
 - Inconsistencies noticed
- Must plan what to do in response
 - Evaluate the situation
- Steps take parameters
 - Different parameters define invocation contexts
- Rework can happen again in the rework context
 - Response to trigger determined by context

A Rework Pattern

- Work Task
 - Parameterized
- Trigger
 - Consistencies
- Rework Task
 - Evaluation
 - Plan
 - Context construction
 - Context
 - Control how re-invocation behaves
 - Re-invocation

Future Work

- What is context and how can it be altered?
 - Parameters?
 - Exception handling?
 - Resources used?
- Supporting evaluation
 - How to manage history?
- Combining with other patterns
 - e.g. Delayed invocation, Exception handling
- Implementation choices
 - Imperative, rule-based languages