Agile Software Development: Software Change

Václav Rajlich, Christopher Dorman
Department of Computer Science
Wayne State University
Detroit, MI 48202
rajlich@wayne.edu  crdorman@gmail.com
Agile practices

**Code production - craftsmanship**
- Shared by all agile processes
- Examples: Refactoring, coding standard, ...

**Team internal practices**
- Organization of work within the team
- Examples: Daily meeting, collective ownership, ...

**Externally visible practices**
- Examples: Small releases, planning game, ...
Software Change (SC)

**Types**

- Perfective
- Adaptive
- Corrective
- Protective

**Most common tasks**

- Fix a bug
- Add a new feature
- Refactor code
- Upgrade GUI
- Add persistent data
- Remove dead code
- Upgrade framework version
- Port to another OS
Wish list: How to change large application?

Quickly
  • Increased velocity

Use only as-needed knowledge
  • Increased velocity and applicability

With minimal risk
  • Better resulting quality

Keep simple design
  • Future evolvability
Phased model of SC

Phases of SC

- Interactions with the world
  - SC design
  - SC Implementation

- Impact Analysis
- Prefactoring
- Actualization
- Postfactoring
- Conclusion

© 2014 by Václav Rajlich
SC starts by a change request

- Elicitation
- Analysis
- Prioritization
- etc.
Hands-on Example

Point of Sale (PoS)

• Supports a small store
• Controls cash register
• Prints sales receipts
• Keeps data about the cashiers
• Controls the inventory

Developed in several steps

• PoS 1.0, PoS 2.0, ...

Code at: http://www.cs.wayne.edu/~vip/PoS/PoS.zip
Change Requests

PoS 2.0
1. Add ability to set a price that takes effect in the future.
2. Add two separate store locations.
3. Add ability to sell items in quantities less than 1, i.e. 0.5.

PoS 5.0
1. Add multiple cashier capability.
2. Log transactions.
3. An error appears and the sale is denied if customer pays with exact change.
Concept Location

Concepts are extracted from change request

Extracted concepts are located in the code and used as a starting point of SC
Impact Analysis

Determine strategy and impact of change

Classes identified in concept location make up the initial *impact set*

Class dependencies are analyzed, and impacted classes are added to the impact set
Concept triangle

- **Name**
  - naming
  - definition

- **Intension**
  - recognition

- **Extension**
  - location
  - traceability

- annotation
Spelling corner

Intension \in-	ext{’}ten(t)-\text{shən}\ 

synonym CONNOTATION

- the suggesting of a meaning by a word apart from the thing it explicitly names or describes b: something suggested by a word or thing — W. R. Inge> an essential property or group of properties of a thing named by a term in logic

Intention \in-	ext{’}ten(t)-\text{shən}\ 

synonyms INTENT, PURPOSE, DESIGN, AIM, END, OBJECT, OBJECTIVE, GOAL mean what one intends to accomplish or attain.

- INTENTION implies little more than what one has in mind to do or bring about
  <announced his intention to marry>. . .

Source: Merriam-Webster
Dog as an example

<<name>>
Dog / Pes / Hund

<<intension >>
Hairy animal with teeth…

<<extensions >>

Fido
Lassie
Buck (in “Call of the wild” by Jack London)
Role of concept location

Concept location finds code snippet where a change is to be made

Change requests are most often formulated in terms of domain concepts

• Example: “Correct error that arises when trying to paste a text”
• the programmer must find in the code the locations where concept “paste” is located
• this is the start of the change
Change Requests

PoS 2.0

1. Add ability to set a price that takes effect in the future.
2. Add two separate store locations.
3. Add ability to sell items in quantities less than 1, i.e. 0.5.
Concept Locations

PoS 2.0

1. Add ability to set a price that takes effect in the future.
   • Answer
2. Add two separate store locations.
   • Answer
3. Add ability to sell items in quantities less than 1, i.e. 0.5.
   • Answer
Partial comprehension of a code

Large programs cannot be completely comprehended
  • programmers seek the minimum essential understanding for the particular software task
  • they use an as-needed strategy
  • they attempt to understand how certain specific concepts are reflected in the code

Analogy: visiting a large city
Formulating a query

Extract the set of concepts used in the change request
Delete the concepts intended for the communication with the programmers
Delete the concepts that are unlikely to be implemented in the code
  • concepts related to the things that are outside of the scope of the program
  • concepts that are to be implemented in the future.
Rank the remaining concepts by the likelihood that they can be easily located
Example

Point of Sale system
Change request is “Implement a credit card payment”
Identify the concepts
- “Implement”
- “Credit card”
- “Payment”
Example

Point of Sale system
Change request is “Implement a credit card payment”
Identify the concepts

- “Implement” ... communication with programmer
- “Credit card”
- “Payment”
Example

Point of Sale system
Change request is “Implement a credit card payment”
Identify the concepts

- “Implement” ... communication with programmer
- “Credit card” ... to be implemented, not in the old code
- “Payment”
Concept location methodologies

Human knowledge

Traceability tools

Dynamic search (execution traces)

Static search

- dependency search
- "grep" (pattern matching)
- information retrieval techniques
Comparison of the Techniques

The grep-based

- depend on the use of naming conventions
- independent of class structure
- GREP tools provide just the list of search results
- suitable for explicit concepts only

The static dependency search technique

- utilizes the class structure
- needs correct understanding of composite and local functionality
- Suitable for both explicit and implicit concepts
### Status of components (marks)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The class was never inspected and is not scheduled for an inspection.</td>
</tr>
<tr>
<td>Propagating</td>
<td>The programmers inspected the class and found that it is impacted by the change.</td>
</tr>
<tr>
<td>Unchanged</td>
<td>The programmers inspected the class and found that it is not impacted by the change.</td>
</tr>
<tr>
<td>Next</td>
<td>The class is scheduled for inspection.</td>
</tr>
<tr>
<td>Located</td>
<td>Concept Location</td>
</tr>
</tbody>
</table>

© 2014 by Václav Rajlich
Locating figure properties: Start

- **UMLEditor**
  - **SequenceDiagram**
    - **ClassRelationship**
      - **MultiLineString**
  - **UseCaseDiagram**
    - **PackageNode**
      - **RectangularNode**
  - **ClassDiagram**
    - **ClassNode**
    - **AbstractNode**
  - **ObjectDiagram**
    - **InterfaceNode**
  - **StateDiagram**
    - **NoteNode**

© 2014 by Václav Rajlich
Most likely supplier

UMLEditor

- SequenceDiagram Graph
- UseCaseDiagram Graph
- ClassDiagram Graph
- ObjectDiagram Graph
- StateDiagram Graph

- ClassRelationship Edge
- PackageNode
- ClassNode
- InterfaceNode
- NoteNode

- MultiLineString
- RectangularNode
- AbstractNode
Next classes to inspect

- SequenceDiagram
  - Graph

- UseCaseDiagram
  - Graph

- ClassDiagram
  - Graph

- ObjectDiagram
  - Graph

- StateDiagram
  - Graph

- ClassRelationship
  - Edge

- PackageNode

- ClassNode

- InterfaceNode

- NoteNode

- MultiLineString

- RectangularNode

- AbstractNode

© 2014 by Václav Rajlich
Wrong way
Concept found

UMLEditor

- SequenceDiagram
  - Graph
- UseCaseDiagram
  - Graph
- ClassDiagram
  - Graph
- ObjectDiagram
  - Graph
- StateDiagram
  - Graph

- ClassRelationship
  - Edge
- PackageNode
- ClassNode
- InterfaceNode
- NoteNode

- MultiLineString
- RectangularNode
- AbstractNode

© 2014 by Václav Rajlich
Change Requests

PoS 5.0

1. Add multiple cashier capability.
2. Log transactions.
3. An error appears and the sale is denied if customer pays with exact change.
PoS 5.0

1. Add multiple cashier capability.
   • Answer

2. Log transactions.
   • Answer

3. An error appears and the sale is denied if customer pays with exact change.
   • Answer
Impact Analysis

Determine strategy and impact of change

Classes identified in concept location make up the initial *impact set*

Class dependencies are analyzed, and impacted classes are added to the impact set

© 2014 by Václav Rajlich
Impact Analysis

What is the simplest strategy?
What can influence IA?
What are technical challenges?
Hard question: When is the right time to stop?
Example: Point of Sale

- Store
- Inventory
- Cashiers
  - CashierRecord
- Item
- Price
Change request

Record cashier sessions

A cashier session

- total cash and all sales
- during the time between the cashier logging in and out.
### Status of components (marks)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The class was never inspected and is not scheduled for an inspection.</td>
</tr>
<tr>
<td>Changed</td>
<td>The class is impacted by the change</td>
</tr>
<tr>
<td>Propagating</td>
<td>The class is not impacted, a client or supplier of the class is impacted.</td>
</tr>
<tr>
<td>Unchanged</td>
<td>The class is inspected but not impacted by the change.</td>
</tr>
<tr>
<td>Next</td>
<td>The class is scheduled for inspection</td>
</tr>
</tbody>
</table>
Example: Point of Sale

- Cashiers
  - CashierRecord
- Store
- Inventory
  - Item
  - Price
Example: Point of Sale

- Cashiers
  - CashierRecord

- Store

- Inventory

- Item
  - Price
Example: Point of Sale
Example: Point of Sale
Alternatives in software change

Program displaying a temperature in Fahrenheit
  • change request: display it in Celsius

Two separate locations deal with temperature
  • sensor data converted to the temperature
  • temperature displayed to the user

The change can be done in either place
  • impact analysis weights these alternatives
The Criteria

Required effort of the change

Clarity of the resulting code

Often, these two criteria contradict each other
  • it is easier to adjust the user interface
  • it is better to have all calculations of the temperature in one place

Conflict between short-term and long-term goals
Change Requests

PoS 2.0
1. Add ability to set a price that takes effect in the future.
2. Add two separate store locations.
3. Add ability to sell items in quantities less than 1, i.e. 0.5.

PoS 5.0
1. Add multiple cashier capability.
2. Log transactions.
3. An error appears and the sale is denied if customer pays with exact change.
Impact Analysis

PoS 2.0

1. Add ability to set a price that takes effect in the future.
   • Answer

2. Add two separate store locations.
   • Answer

3. Add ability to sell items in quantities less than 1, i.e. 0.5.
   • Answer
Impact Analysis

PoS 5.0

1. Add multiple cashier capability.
   • Answer

2. Log transactions.
   • Answer

3. An error appears and the sale is denied if customer pays with exact change.
   • Answer
Why bother? Ariane 5
Prefactoring

Opportunistic refactoring that localizes (minimizes) impact of SC on software

*Extract Class Component*
  - gather fields, methods, and code snippets into a new component class

*Extract Superclass*
  - create new abstract class
Other examples of refactoring

Rename an entity

Encapsulate part of the code as a function
  • opposite: expand a function in a place of call

Move a member function from one class into another

Merge and divide classes
  • factor out a base class, component class
Change Requests

PoS 2.0
1. Add ability to set a price that takes effect in the future.
2. Add two separate store locations.
3. Add ability to sell items in quantities less than 1, i.e. 0.5.

PoS 5.0
1. Add multiple cashier capability.
2. Log transactions.
3. An error appears and the sale is denied if customer pays with exact change.
Prefactoring

PoS 2.0

1. Add ability to set a price that takes effect in the future.
   • Answer

2. Add two separate store locations.
   • Answer

3. Add ability to sell items in quantities less than 1, i.e. 0.5.
   • Answer
Prefactoring

PoS 5.0

1. Add multiple cashier capability.
   • Answer

2. Log transactions.
   • Answer

3. An error appears and the sale is denied if customer pays with exact change.
   • Answer
Actualization

Creates new code
Plugs it into the old code
Visit neighboring classes and update them
  • Change propagation (ripple effect)
Postfactoring

Eliminate any anti-patterns that may have been introduced

- long method
  - after added functionality, some methods may be doing too much
- bloated class
  - after added functionality, a class may be too large
SC Conclusion

- Commit finished code into version control
- Build the new baseline
- Release?
- Prepare for the next change
Web site from SourceForge

JRipples
Brought to you by: cdorman, petrenkomaxim, raduvanciu, xrajlich, ybinw

Description

JRipples is an Eclipse plug-in that supports a programmer during software maintenance and evolution in two ways: it does relevant program analysis, and it manages organization of steps that comprise the impact analysis and subsequent change propagation.
Conclusions

Software change is the most common task

Phased model of software change

- Increase in productivity
  - As needed knowledge
  - Well-defined process
- Better quality
  - Fewer residual bugs
- Better evolvability
  - Preserve simple design
Learn More

Software Engineering
The Current Practice

Václav Rajlich

© 2014 by Václav Rajlich