8 Actualization

- Programmers implement the new functionality
  - according to change request
- The process of actualization varies
  - depends on the size of the change
Small changes

• Done directly in old code

class Address
{
    public move();
    protected String name;
    protected String streetAddress;
    protected String city;
    protected char state[2], zip[5];
};
Small changes

• Done directly in old code

class Address
{
    public move();
    protected String name;
    protected String streetAddress;
    protected String city;
    protected char state[2], zip[9];
};
Small changes

- Done directly in old code

class Address
{
    public move();
    protected String name;
    protected String streetAddress;
    protected String city;
    protected char state[2], zip[9];
};
Larger changes

• Programmers implement the new classes separately from the old code
• The new code is plugged into the existing code
  – incorporation
• The change can propagate to other components of the system
  – ripple effect
Polymorphism

Old Code

Farm
Cow
Sheep
FarmAnimal

New Code

Pig
Polymorphic class

class Pig : public FarmAnimal
{
    public:
    void makeSound() {cout<<"Oink";}
};

• Farm now can declare objects of the type Cow, Sheep, or Pig
  – the composite responsibility of Farm was extended by the concept Pig.
Adding New Component

• Implementation
  – New component is implemented separately from the clients in the old code
  – new classes assume the responsibilities demanded by the change request

• Incorporation
  – New classes are plugged as components into the appropriate composite class of the existing code
New component implementation
New component incorporation
Change propagation

- After component incorporation, change propagates to the neighbors of the composite
- Process is step-by-step
  - Similar to impact analysis
  - This time, real changes in the code are made
Point of Sale

• The old application did not compute the tax
• The change request:
  – Add tax computation to Point of Sale application
Point of Sale
Example incorporation

- register
- store
- item
- sale
- saleLineItem
- tax
Example incorporation
Change propagation

- register
- sale
- store
- item
- saleLineItem
- taxCategory
Change propagation

- register
- store
- item
- sale
- saleLineItem
- taxCategory
Change propagation

- register
- store
- item
- sale
- saleLineItem
- taxCategory
Change propagation

- register
- store
- item
- sale
- saleLineItem
- taxCategory
Change propagation ends
New functionality is a composite
Incorporating new functionality

Old Code

New Code

© 2012 Václav Rajlich
Replacement of a class

© 2012 Václav Rajlich            Software Engineering: The Current Practice   Ch. 8
Deletion of obsolete functionality

• Also causes change propagation
• All references to the deleted functionality must be deleted
  – secondary changes propagate to other classes
Underestimated Impact Set

- Impact analysis estimates which classes are impacted
- Change propagation modifies the code of impacted classes
  - change propagation is the moment of truth
  - it confirms or refutes the predictions of impact analysis
  - accuracy of impact analysis predictions is important for software managers
Ericsson Radio Systems

<table>
<thead>
<tr>
<th></th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unchanged</td>
</tr>
<tr>
<td>Actual</td>
<td></td>
</tr>
<tr>
<td>Unchanged</td>
<td></td>
</tr>
<tr>
<td>Changed</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>42</td>
</tr>
</tbody>
</table>

• total number of classes =
  
  \[42 + 0 + 64 + 30 = 136\]
Categories

- true positives = 30
- false positives = 0
- true negatives = 42
- false negatives = 64
Precision

• Used in the information retrieval
• Precision = \( \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \)
• Ericson, precision = \( \frac{30}{30 + 0} = 1 = 100\% \).
Recall

• Recall = (true positives)/(true positives + false negatives)
• Ericson recall = 30/(30 + 64) = 0.32 = 32%
• Programmers estimated that the changes will impact only about a third of all classes that actually changed
  – missed the other two thirds!
Underestimation

- Common in software engineering
  - consequence of invisibility
- Makes planning difficult
- Common in other field also