5 Introduction to software change

• Software change (SC) is the process of adding new functionality to existing code

• Foundation of software evolution, servicing
Characteristics of SC

- Lientz and Swanson
  - perfective ~66%
  - adaptive
  - corrective
  - protective
Functionality

• Incremental
  – adding new functionality

• Contraction
  – removing obsolete functionality

• Replacement
  – replacing existing functionality

• Refactoring
  – changing software structure without changing behavior
Impact

- Local impact
- Significant impact
- Massive impact

- Change strategy
  - improves structure
  - quick fix
Form of changing code

• Source code
  – most common

• Code after compilation
  – object form
  – executable form
Phased model of SC

- Main topic of this course
- Preview of Phases of SC
Initiation

- SC starts by a change request
- Prioritization of change requests, etc.
SC Design

- Concept
- Location
- Impact Analysis
- Prefactoring
- Actualization
- Postfactoring
- Conclusion
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 \textit{impact set}
• Class dependencies are analyzed, and impacted classes are added to the impact set
Prefactoring

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

• *Extract Class* (Fowler)
  – gather fields, methods, and code snippets into a new component class

• *Extract Superclass*
  – create new abstract class
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
Verification

• Guarantees correctness of the change
• Testing
  – functional
  – unit
  – structural
• Walkthroughs
Conclusion

• Commit finished code into version control
• Build the new baseline
• Prepare for the next change
Test-Driven Development

- Write test first
- Write code to pass the test
Change initiation

- Requirements
  - user reports a software bug
  - user asks for an enhancement
  - programmer proposes improvement
  - manager wants to meet competitor’s functionality
Requirements form

• Sentence or paragraph
• Bug report
• User story
  – limit the complexity of the story and potential for misunderstanding
  – user story fits on a 3” x 5” card
  – if a new functionality cannot fit, it has to be divided into several user stories
Product backlog

• Database of requirements
  – “wish list”
  – add/delete/modify requirements
  – additional knowledge is acquired by the users
  – additional clarification is needed by the developers
Requirements Elicitation

Stakeholders → Requirements elicitation → New Requirements

Product Backlog

Desired Code

Existing Code
Requirements analysis

• Inconsistencies
  – Contradictions
    • ex. different formula for the same thing
  – Inadequacy
    • ex. requirements are too terse -> developers have to guess
Inconsistencies (2)

- Noise
  - ex. irrelevant requirements (delete them)
- Unfeasibility
  - ex. project team or technology barriers
- Ambiguity or unintelligibility
  - ex. interpreting a requirement in more than one way
Prioritization - bugs

- 1. Fatal application error
- 2. Application is severely impaired
  - no workaround can be found
- 3. Some functionality is impaired
  - workaround can be found
- 4. Minor problem
  - not involving primary functionality
Business value

• 1. An essential functionality without which the application is useless
• 2. An important functionality that users rely on
• 3. A functionality that users need but can be without
• 4. A minor enhancement
Risk

• 1. A serious threat, the so-called “showstopper”
  – if unresolved, the project is in serious trouble
• 2. An important threat that cannot be ignored
• 3. A distant threat that still merits attention
• 4. A minor inconveniences
Process needs

• 1. Key requirement
  – if not implemented in advance, practically all code will have to be redone
• 2. An important requirement
  – if postponed, will lead to large rework
• 3. A nontrivial rework will be required if this requirement is postponed
• 4. A minor rework will be triggered
Change Initiation process

• Select a set of the highest priority requirements
• Analyze these requirements
• After this analysis, select the highest priority requirement as the next change request
Change Initiation

Stakeholders → Change Initiation → Change Request → Product Backlog → Change → Desired Code

Existing Code → New Code → Desired Code