Software Change (SC)

1. Software change
   1. Software change is the process of adding new functionality to existing code
      1. Foundation of software maintenances, evolution, and agile processes
2. Characteristic of SC
   1. Lientz and Swanson
      1. Perfective: ~66%: new functionality
      2. Adaptive: new environment
      3. Corrective: Fix bugs
      4. Protective: make future change easier
3. Functionality
   1. Incremental: More features?
   2. Contraction: Remove obsolete functionality. Sure? Impacts?
   3. Replacement: replace with better one
   4. Replace buggy code with correct one
   5. Refactoring
4. Impact
   1. Local impact
   2. Significant impact
   3. Massive impact
   4. Change strategy
      1. Improves structure
      2. Quick fix
5. Form of changing code
   1. Source code
   2. After compilation
      1. Object form
      2. Executable form
6. SC process model
   1. Phases of SC
      1. Interactions with the world
         1. SC design
         2. SC implementation
   2. Process
      1. Initiation
      2. Concept location
      3. Impact analysis
      4. Prefactoring
      5. Actualization
      6. Postfactoring
      7. Conclusion
7. Initiation
   1. SC starts by a change requires
   2. Prioritization of change requests
8. SC Design (Concept location, Impact analysis)
   1. Concept location
      1. Concepts are extracted from change request
      2. Extracted concepts are located in the code and used as a starting point of SC
   2. Impact analysis
      1. Determine strategy and impact of change
      2. Classes identified in concept location make up the initial impact set
      3. Class dependencies are analyzed, and impacted classes are added to the impact set
9. Prefactoring
   1. Opportunistic refactoring that localizes (minimizes) impact of SC on software
   2. Extract Class (Fowler)
      1. Gather fields, methods, and code snippets into a new class
   3. Extract Superclass
      1. Create new abstract class
10. Actualization
    1. Creates new code
    2. Plugs it into the old code
    3. Beginning with the class with new code, visit neighboring classes and update them
       1. Change propagation
       2. Ripple effect
11. Postfactoring
    1. Eliminate any anti-patterns that may have been introduced
       1. Long method
          1. After added functionality, some methods may be doing too much
       2. Bloated class
          1. After added functionality, a class may be too large
12. Verification
    1. Guarantees correctness of the change
    2. Testing
       1. Functional
       2. Unit
       3. Structural
       4. Walkthroughs
13. Conculsion
    1. Commit finished code into version control
    2. Build the new baseline
    3. Release
    4. Prepare for next change
14. Test-driven development
    1. Write test first
    2. Write code to pass the test
15. Change initiation: requirements
    1. A user asks for a functionality enhancement
    2. A user reports a software bug
    3. A programmer proposes improvement
    4. A manager wants to meet competitor’s functionality
    5. Perfective, adaptive, corrective, protective
16. Requirements form
    1. A sentence or paragraph
    2. Bug report
    3. User story
       1. Limit the complexity of the story and potential for misunderstanding
       2. User story fits on a 3” x 5” card
       3. If a new functionality cannot fit, it has to be divided into several user stories
17. Product backlog
    1. Database of requirements
    2. Can be prioritized
       1. New stories can be easily added or deleted
       2. Old user stories can be rewritten
       3. Additional knowledge is acquired by the users
       4. Additional clarification is needed by the programmers
    3. Wish list
       1. Tools like buzilla
       2. Add/delete/modify/reprioritize change requests
18. Requirements analysis
    1. Inconsistencies
       1. Contradictions
       2. Inadequacy
       3. Noise
       4. Infeasibility
       5. Ambiguity
    2. Prioritization
       1. Bugs
          1. Fatal application error
          2. Application is server impaired
          3. Some functionality is impaired
          4. Minor problem not involving primary functionality
    3. 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
19. Risk
    1. A serious threat, 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 inconvenience
20. Process needs
    1. A key requirement
       1. If not implemented in advance, practically all code will have to be redone
    2. An important requirement that 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
21. Change Initiation Process
    1. Select a set of the highest priority requirements
    2. Analyze these requirements
    3. After this analysis, select the highest priority requirement as the next change request