Refactoring

1. Refactoring
   1. Refactoring changes the software structure but does not change the functionality of the program
      1. Important activity during evolution
   2. Refactoring consists of behaviour preserving transformation
2. Two roles of refactoring
   1. Opportunistic refactoring
      1. Pre-factoring
      2. Before attempting the actualization
         1. Refactoring will make actualization easier
   2. Improving the architecture
      1. Post-factoring
      2. Prepares software for future evolution
      3. Driven by esthetics, not specific need
3. Examples of refactoring
   1. Rename an entity
   2. Encapsulate part of the code as function
      1. Opposite: expand a function in a place of call
   3. Move a member function from one class into another
   4. Merge and divide classes
4. Extract a base class
   1. In the design process, derived classes always come before the base classes
      1. Designers may miss some base classes
      2. Refactoring base class will correct these omissions
   2. Extract a base class prepares software for incorporation of new functionality through polymorphism
      1. Applicable when old and new functionality have large overlap
5. Refactoring process
   1. Rename class
   2. Refactoring and incorporation (create new super class/abstract class)
   3. Make renamed class as derived from the super class
   4. Move variables
   5. Move functions
6. Result
   1. After refactoring, it is easy to incorporate
   2. Refactoring preserves the behaviour
7. Extract function
   1. During the software evolution, some functions may grow to be too large
   2. Or we may need to separate two concepts the function currently deal with it
   3. Extracting part of the function into another function will make it
      1. Easier to understand
      2. Reusable
8. Extract function process
   1. Select a block of code for extraction
   2. Is the block syntactically complete?
   3. Create new function
   4. Extract the selected block as a function body
   5. Replace the code block with the function call
9. Variables during extract function
   1. Local variable
      1. Value assigned inside, used only inside
   2. Parameter passed by value
      1. Value assigned outside
   3. Parameter passed by reference
      1. Values assigned and used outside, changed inside
   4. Global variable
10. Component class extraction
    1. Motivation: Incorporation by replacement
       1. Primitive implementation of the class is replaced by a full functionality
    2. Concept sometimes does not have class of its own
       1. Must be extracted from another class
       2. Pre-factoring for incorporation by replacement
11. Actions of function insertion
    1. The function’s header is inserted into the class specification
    2. Change access to the members of the target class
    3. The function header must be qualified by the class identifier
    4. The parameter that is now replaced by membership must be removed
    5. All function calls must be qualified with a class instance
    6. All forward declarations of the function are replaced by the new function declaration in the target class specification