Quiz

1. How is the staged model different from traditional models of the software life cycle?

1

Introduction to software change

CS 515, Spring 2020

Laura Moreno
lmorenoc@colostate.edu

2

Software change |ˈsôf(t)wer chānj|

noun

1 *Eick'01* any alteration to the software recorded in the change history data base

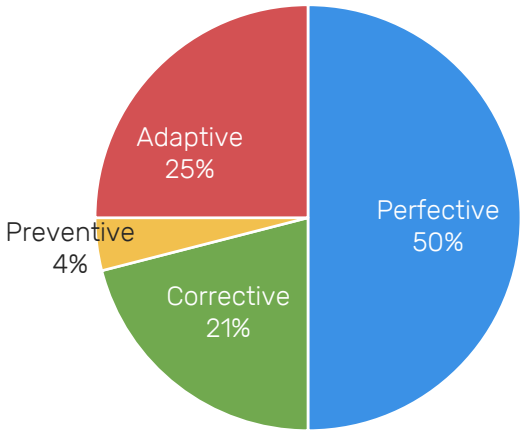
2 the foundation of software maintenance, evolution, and agile processes

3 *Rajlich'00* basic operation of both software evolution and software servicing

4 *Rajlich'00* process of introducing new requirements into an existing system, or modifying the system to fix an existing requirement

3

Purpose of software change



Effort distribution in maintenance
Lientz and Swanson (1980)

Perfective

- Introduce new functionality
- Response to changes in requirements
- Increase the value of software

Adaptive

- Adapt to changes in the operational environment
- Protect the value of the software

Corrective

- Fix software bugs and malfunctions
- Protect the value of the software

Preventive

- Ameliorate performance and maintainability
- Invisible to the user
- Shield the value of the software

4

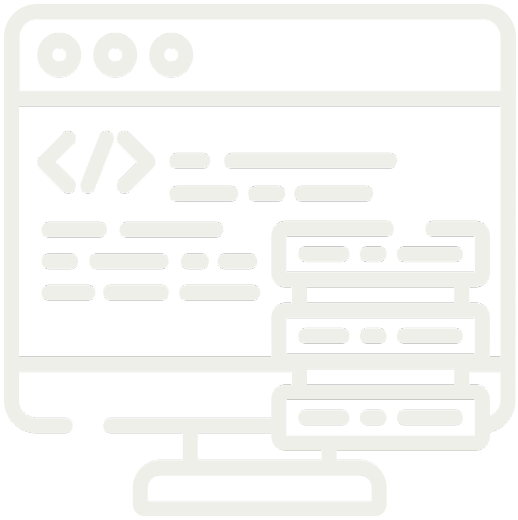
Impact of the change on functionality

Incremental
- Adding new functionality

Contraction (aka code pruning)
- Remove obsolete functionality

Replacement
- Replace existing functionality

Refactoring (or restructuring)
- Change software structure without changing behavior



Colorado State University

@lymorchoc

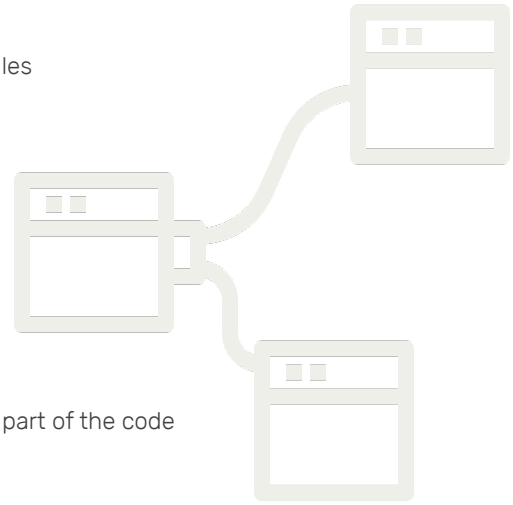
5

Impact of the change in the code

Local impact
- Impact one module or a handful of closely related modules
- Easy to implement
- Consequences are likely to be predictable

Significant impact
- Affect a medium but significant number of modules
- Rarer than small changes, but frequent enough

Massive impact
- Massively delocalized changes that affect a substantial part of the code
- Expensive and risky
- Relatively rare

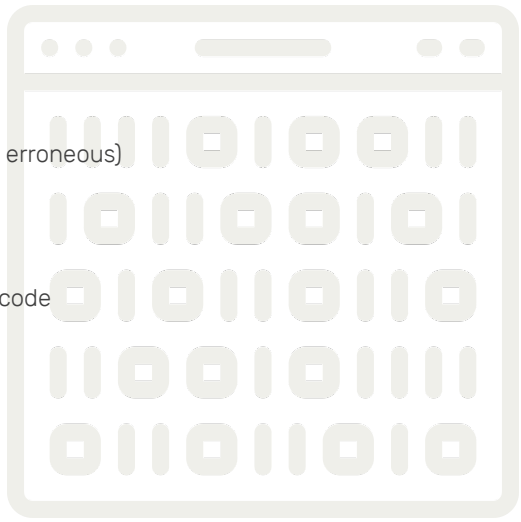


Colorado State University

@lymorchoc

6

Ways of changing code



Executable code

- Unusual circumstances (e.g., code is not accessible)
- Emergencies (compilation takes too long, compilation is erroneous)
- Deal with consequences
  - Human-oriented aspects are gone
  - Code is no longer the most up-to-date artifacts
  - Subsequent changes most be made in executable code

Source code

- Much easier
- Most common and most frequent

7

Change strategy

Quick fix

- Shortcuts might lead to scars
- Only acceptable in critical situations
- Common during servicing

Long-term investment

- Well designed and executed
- Should improve the quality of the code (local and global) and documentation

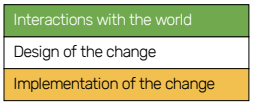
8

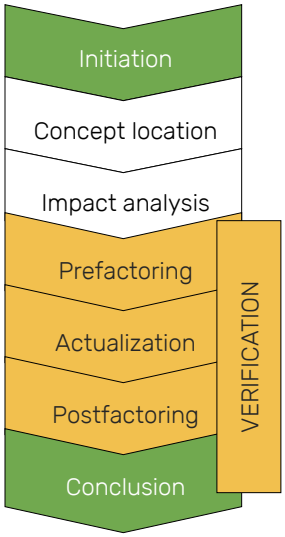
Phases of software change

Building blocks are called phases

A specific software change consist of some combination of these phases

The combination here describes the phases of a typical midsized software change



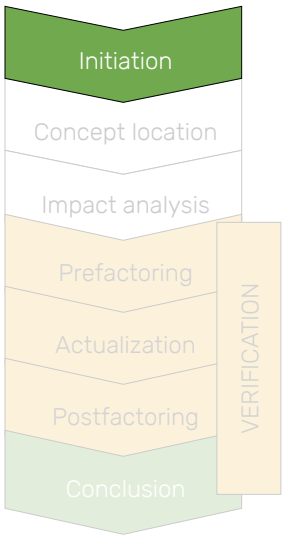


9

Software change >> Initiation

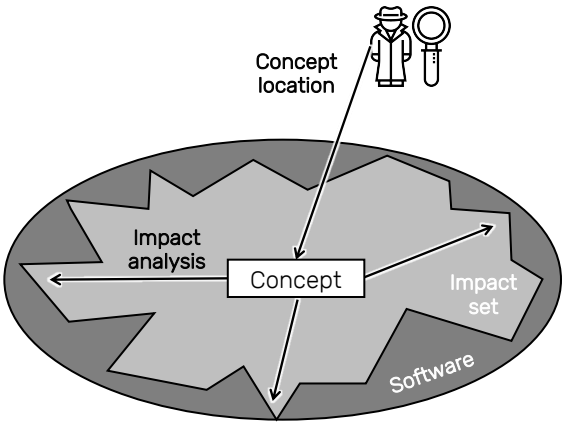
Software change starts with a change request

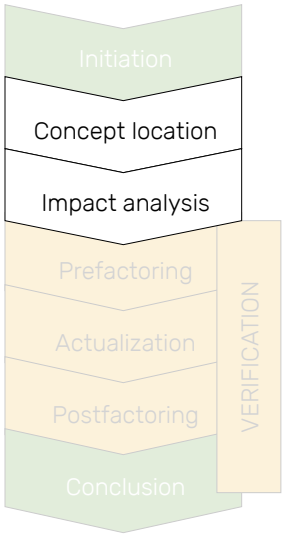
- Prioritization of change requests
- Assigning change request to developers (e.g., bug triage)



10

Software change >> Design





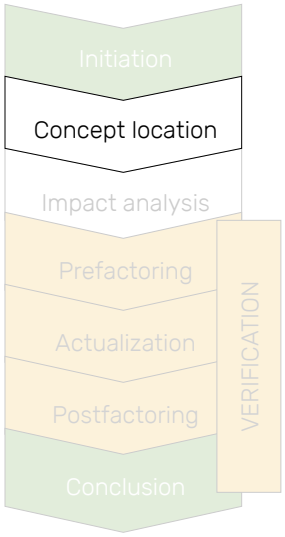
11

Software change >> Concept location

Concepts are extracted from change request

- Change request analysis

Extracted concepts are located in the code and used as a starting point of software change



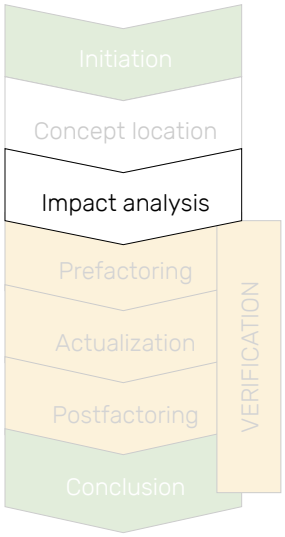
12

Software change >> Impact analysis

Determine the strategy and impact of change

Classes/modules identified in concept location make up the initial impact set

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



13

Software change >> Prefactoring

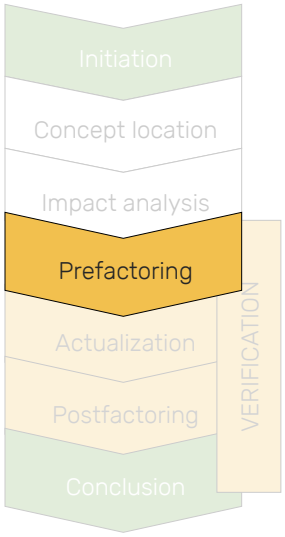
Opportunistic refactoring that localizes (minimizes) impact of software change on software

Extract class

- Gather fields, methods, and code snippets into a new class

Extract superclass

- Create new abstract class



14

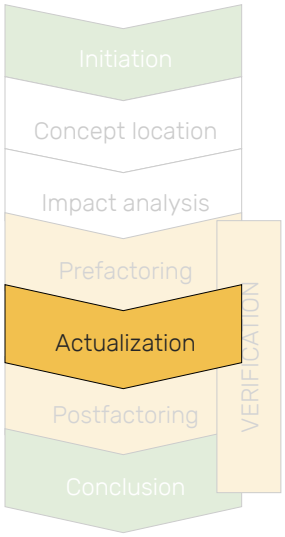
Software change >> Actualization

Create new code

Merge with the existing code

Beginning with the class with new code, visit neighboring classes and update them

- Change propagation
- Ripple effect

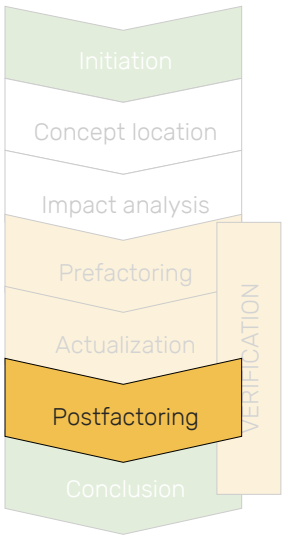


15

Software change >> Postfactoring

Eliminate any anti-patterns and bad smells 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



16

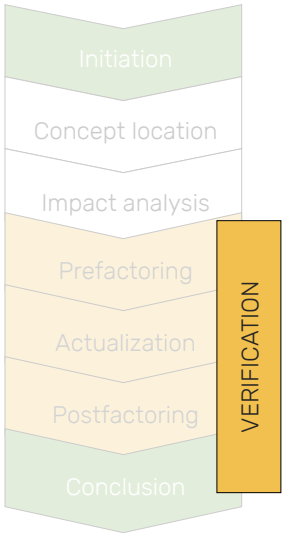
Software change >> Verification

Guarantee correctness of the change

Testing

- Functional
- Unit
- Structural

Walkthroughs



17

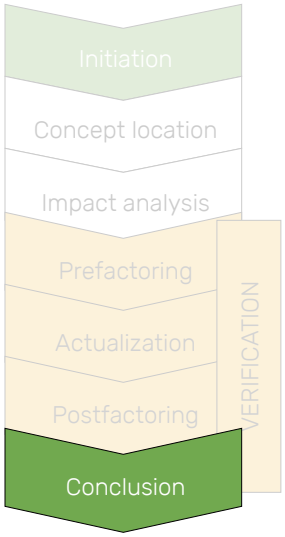
SC >> Conclusion

Commit finished code into the version control system

Build the new baseline

Release?

Prepare for the next change



18

Test-driven development

Write test first

Write code to pass the test