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

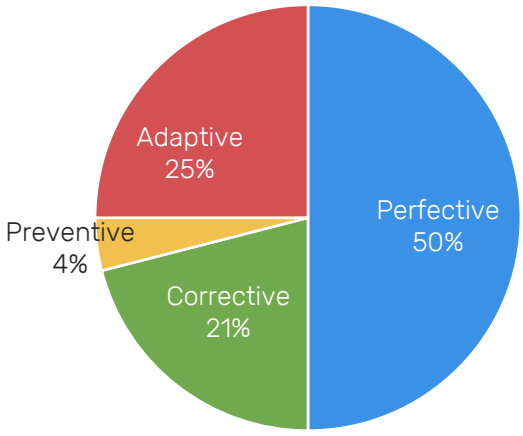
2 the foundation of software maintenance, evolution, and agile processes

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

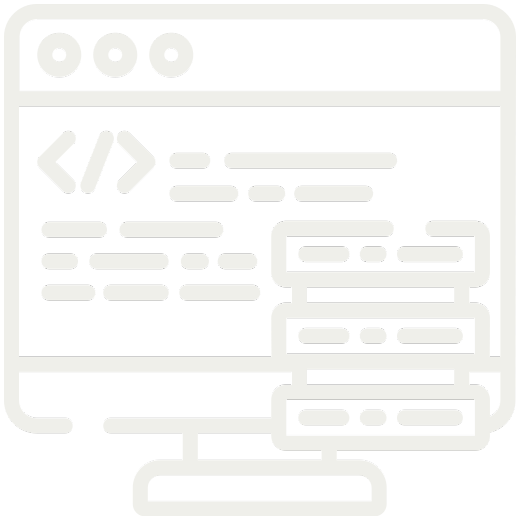
Impact of the change on functionality

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

@lymorchoc

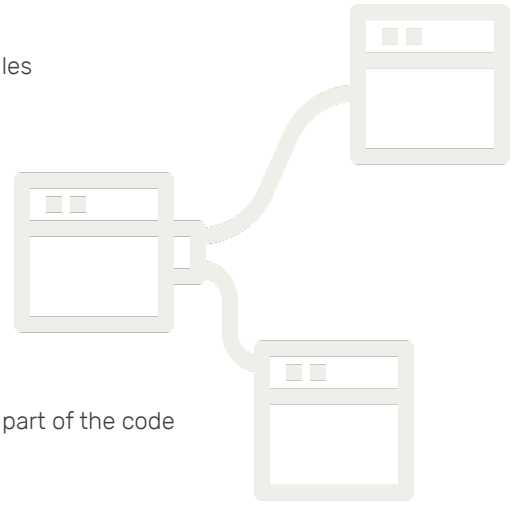
5

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

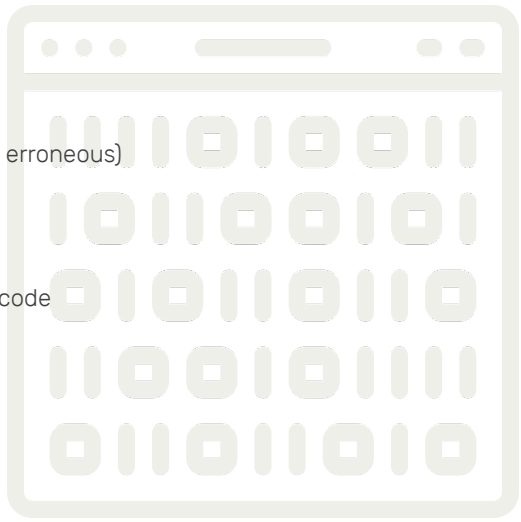


Colorado State

@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

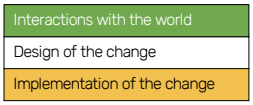
8

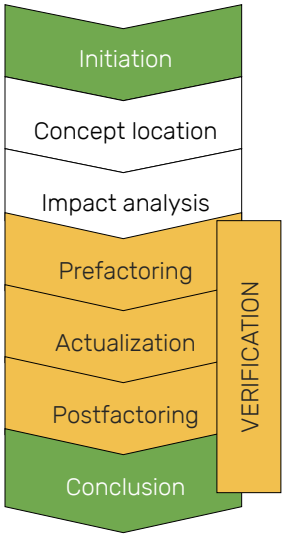
Phases of software change

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



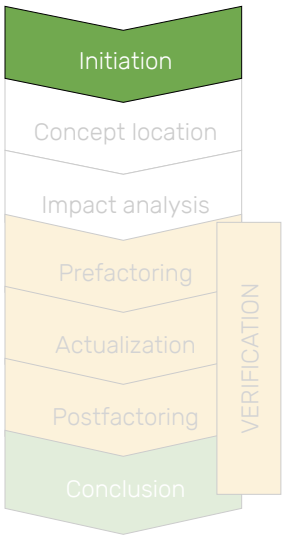


9

Software change >> Initiation

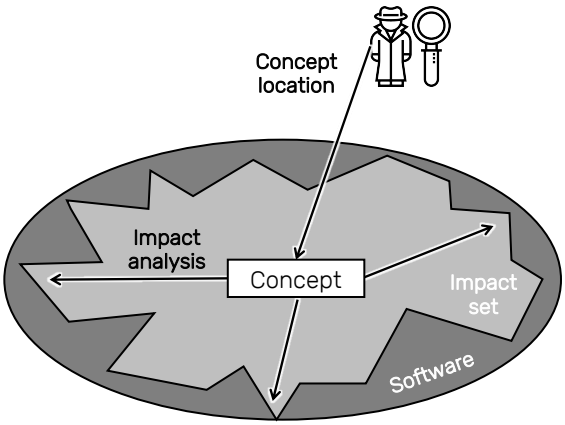
Software change starts with a change request

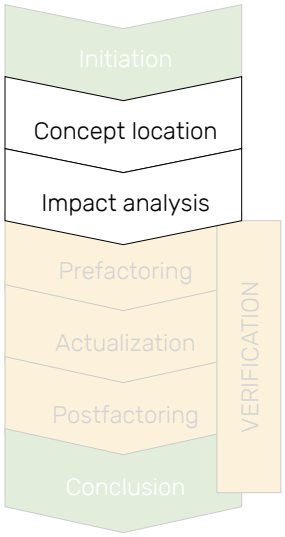
- Prioritization of change requests
- Assigning change request to developers (e.g., bug triage)



10

Software change >> Design





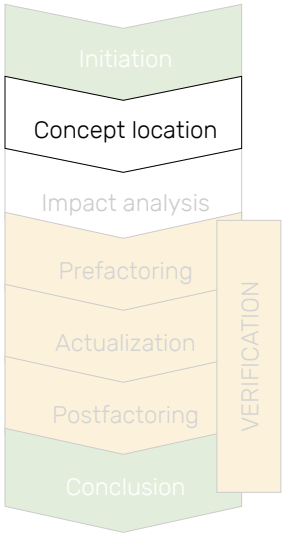
11

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



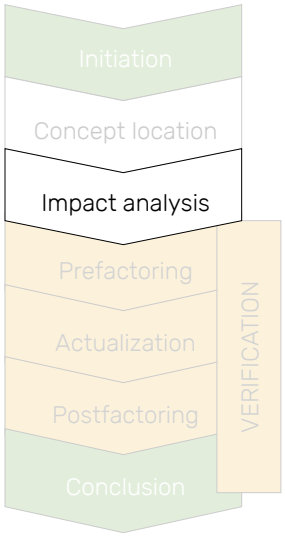
12

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



Software change >> Prefactoring

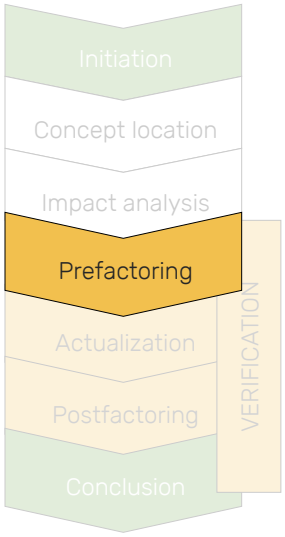
Opportunistic refactoring that localizes (minimizes) impact of software change on software

Extract class

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

Extract superclass

- Create new abstract class



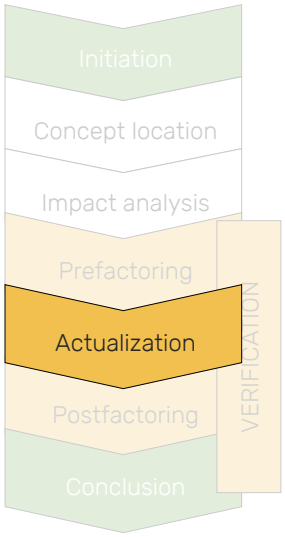
Software change >> Actualization

Create new code

Merge with the existing code

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

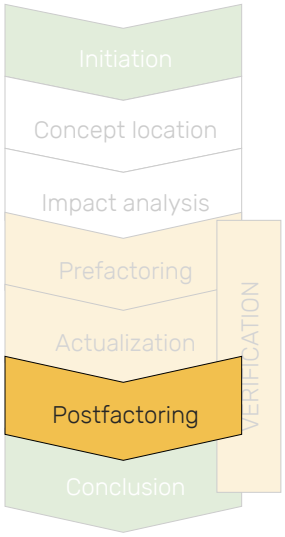
- Change propagation
- Ripple effect



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



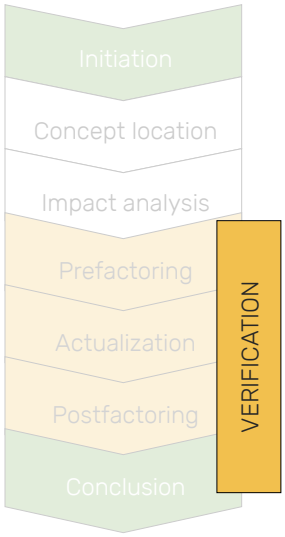
Software change >> Verification

Guarantee correctness of the change

Testing

- Functional
- Unit
- Structural

Walkthroughs



17

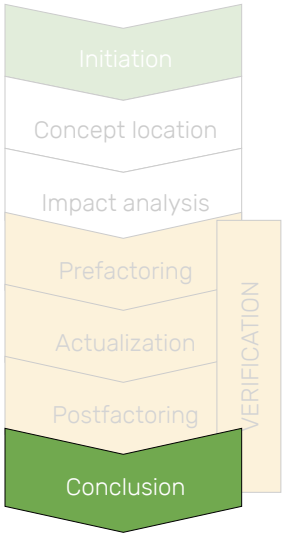
SC >> Conclusion

Commit finished code into the version control system

Build the new baseline

Release?

Prepare for the next change

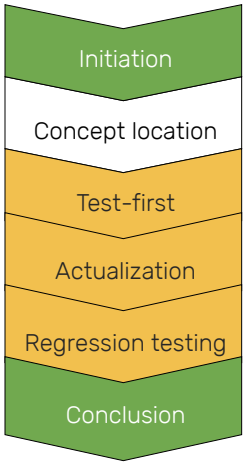


18

Test-driven development

Write test first

Write code to pass the test



19

Concept location in software

20

Concept location in software change

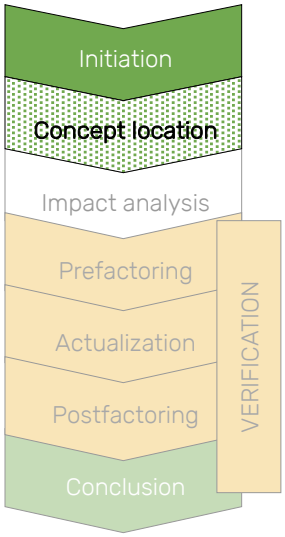
Concept location is part of the software change process

Input:

- The result of initiation

Output:

- Location of change



The role of concept location

Concept location is needed whenever a change is to be made

Change requests are most often formulated in terms of domain concepts

- Example: "*Correct error that arises when trying to paste a text*"
- The programmer must find in the code the locations where concept "paste" is located
- This is the start of the change

Partial comprehension of source code

Large programs cannot be completely comprehended

- Programmers seek the minimum essential understanding for the software task at hand
- They use an as-needed strategy
- They attempt to understand how certain specific concepts are reflected in the code

Analogy: visiting a large city

Concept triangle

Name

Intension

Spelling corner

Intension | inˈtenSHən |

synonym CONNOTATION

1 the suggesting of a meaning by a word apart from the thing it explicitly names or describes

2 something suggested by a word or thing

3 an essential property or group of properties of a thing named by a term in logic

Intention | inˈten(t)SH(ə)n |

synonyms INTENT, PURPOSE, DESIGN, AIM, END, OBJECT, OBJECTIVE, GOAL mean what one intends to accomplish or attain

1 what one intends to do or bring about

Colorado State

@lymorchoc

25

Concept triangle

e.g., "cat"

Name

e.g., Chloe, Ivy

e.g., "a small domesticated carnivorous mammal with soft fur, a short snout, and retractable claws."

Intension

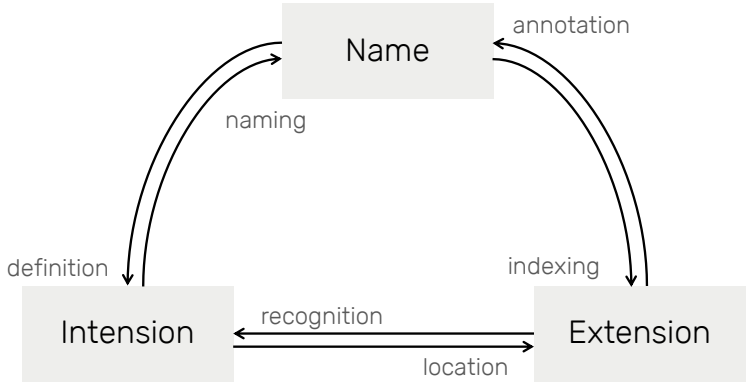
Extension

Colorado State

@lymorchoc

26

Concept triangle



27

Concept location

The extensions of concepts are implemented as code fragments

- variables, classes, methods, or other

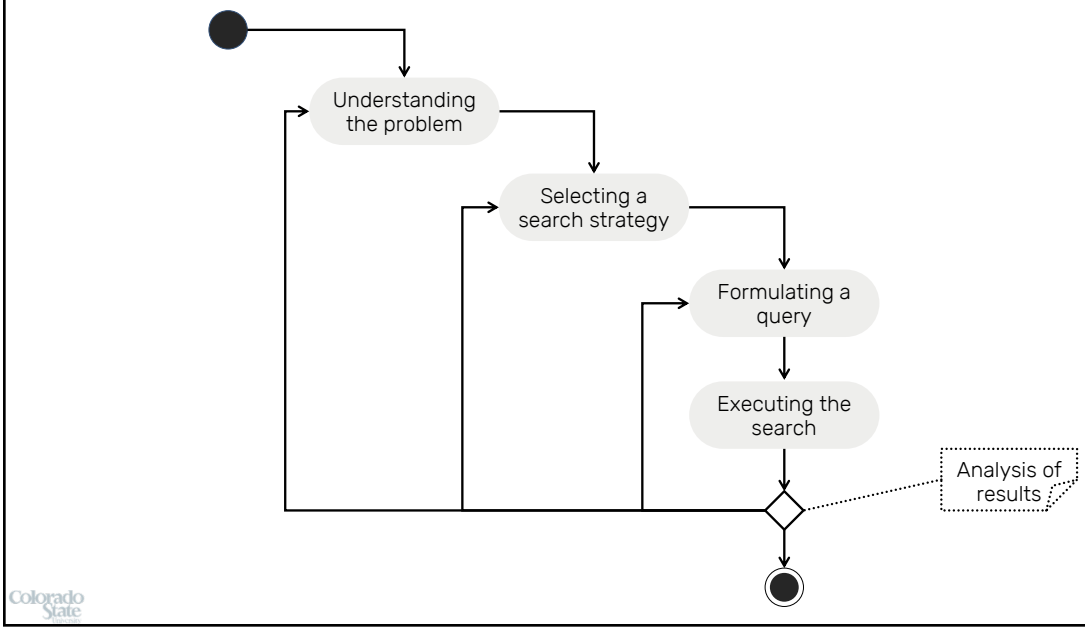
Concept location is a search for specific fragments

This task is:

- easy in small programs or in programs that the programmer knows well
- hard in large programs or programs that the programmer does not know

28

Concept location as a search process



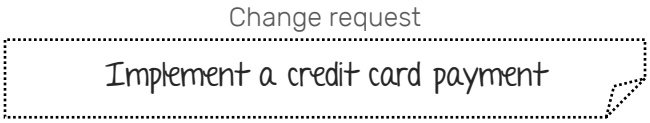
29

Formulating a query

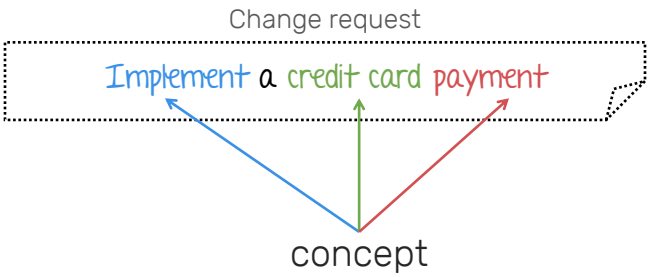
1. Extract the set of concepts used in the change request
2. Delete concepts intended for communication with programmers
3. Delete concepts that are unlikely to be implemented in the code
   - concepts related to things outside of the scope of the program
   - concepts that are not yet implemented
4. Rank the remaining concepts by the likelihood that they can be easily located

30

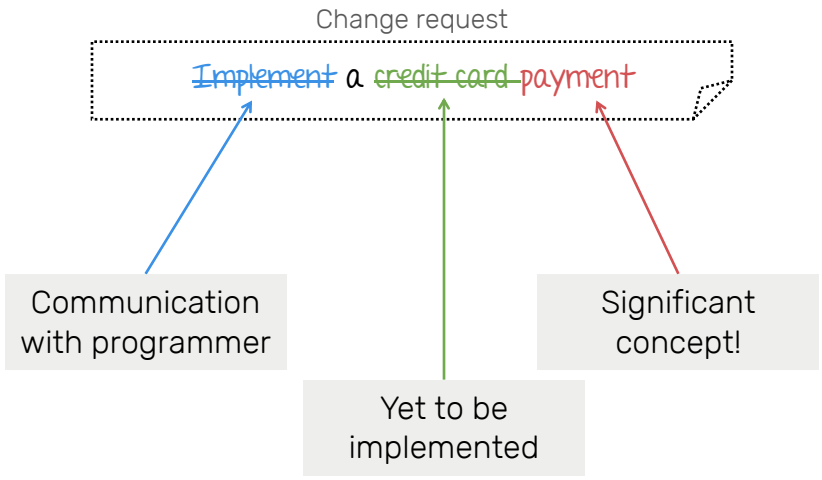
Example ::: Point-of-sale system



Example ::: Point-of-sale system



Example ::: Point-of-sale system



33

Concept location methodologies

Human knowledge

Traceability tools

Dynamic search (execution traces)

Static search
- text search
- dependency search

34

Concept location using text search

Classical technique for concept location

- Based on pattern matching

Steps

1. Programmer formulates a query based on concept name(s)
2. The search engine searches the files and finds corresponding lines of code ("hits")
3. Programmer investigates the hits
   a. If a search fails, new query is tried (programmer learns from failed search)

35

Text searching with `grep`

`grep` is an acronym for "global regular expression print"

- `grep` prints out the lines that contain a match for a regular expression – popular UNIX script for regular expression search
- Programmer iteratively formulates search query and then investigates the results
- If the results are too big to review, programmer either performs further search within these results or reformulates the search query

IDEs support such type of search

Text retrieval is more sophisticated—used for repository search

36

Concept location using dependency search

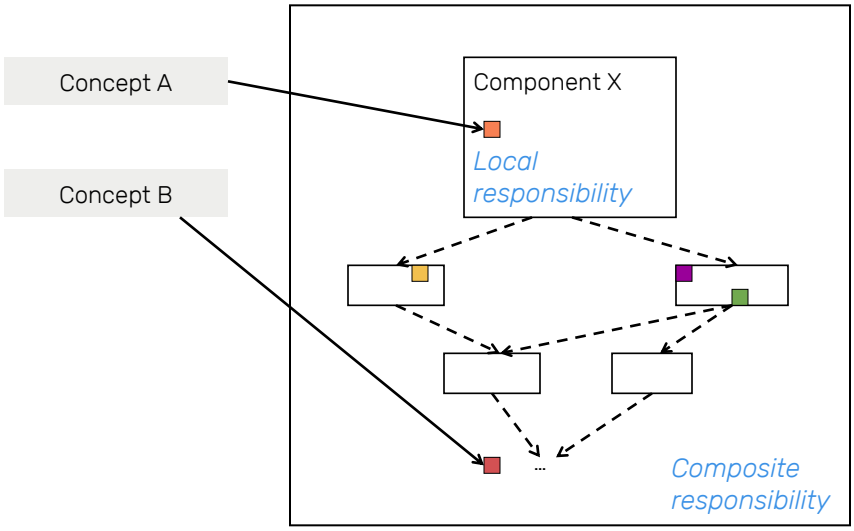
Uses Class Dependency Graphs (CDG)

Local functionality
- consists of concepts that are actually implemented in the module and are not delegated to others.

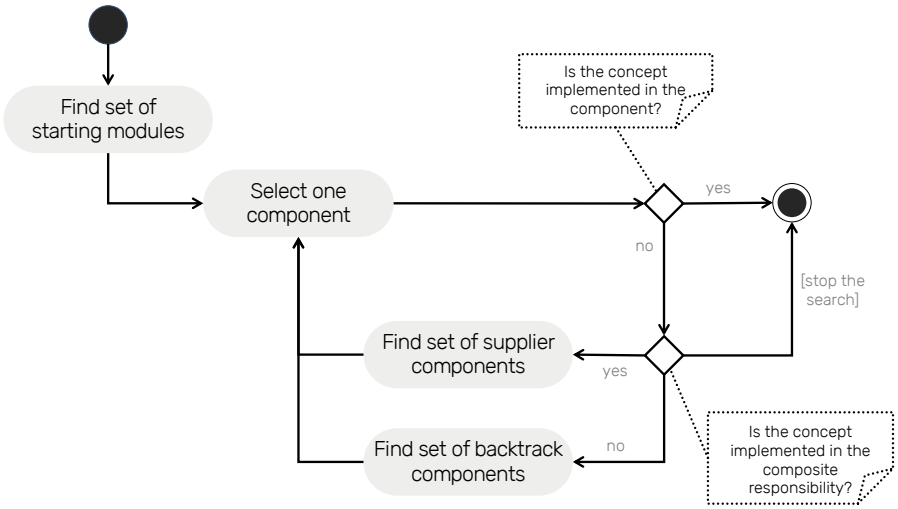
Composite functionality
- as the complete functionality of a module combined with all its supporting modules.

Determined by reading code and documentation

Functionalities of component X



Concept location via dependency search



39

Example ::: Violet

Violet

- Open source UML editor

Supports drawing UML Diagrams

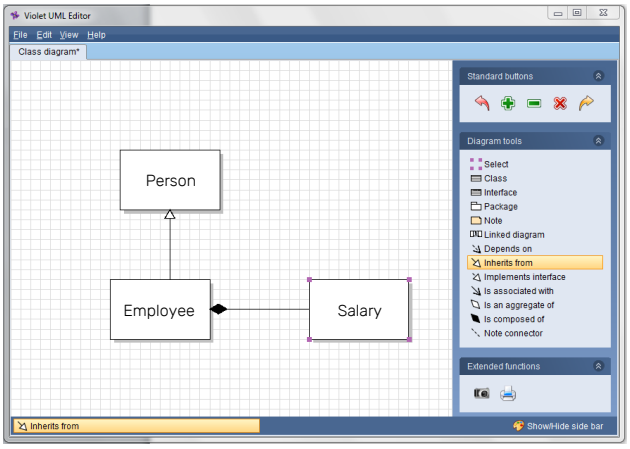
- Class diagram, Sequence diagram, State diagram, Object diagram, Use case diagram

60 classes and 10,000 lines of code

- http://sourceforge.net/projects/violet/

40

GUI of Violet



Change request

Record the author for each class symbol
in the class diagrams

This change will make Violet more versatile

- Support for cooperative work, as the author who created a figure knows its semantics

Change request analysis

Record the author for each class symbol
in the class diagrams

"*record*" – communication

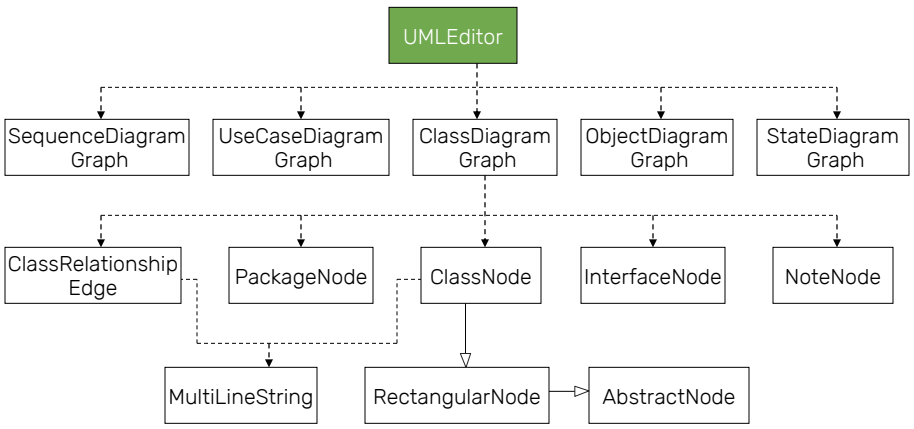
"*author*" – implicit extension (not implemented yet, i.e., extension not present in the code)

"*class diagram*" – too generic

"*class symbol*" – the significant concept to be located

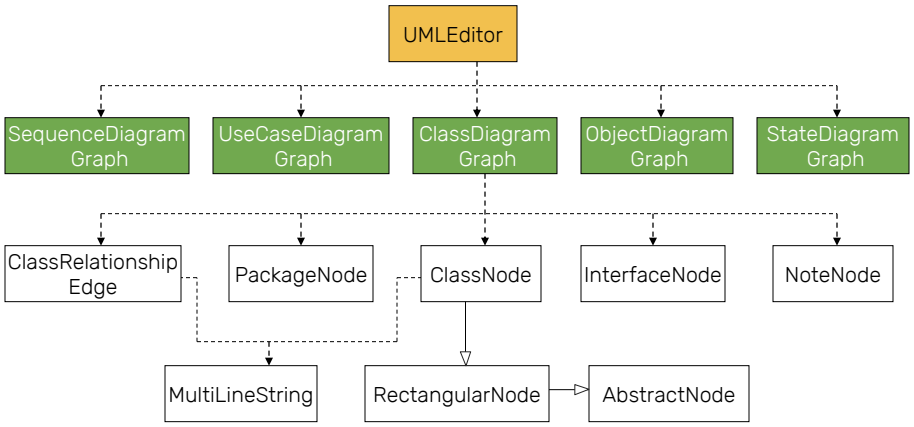
43

Locating figure properties: Start



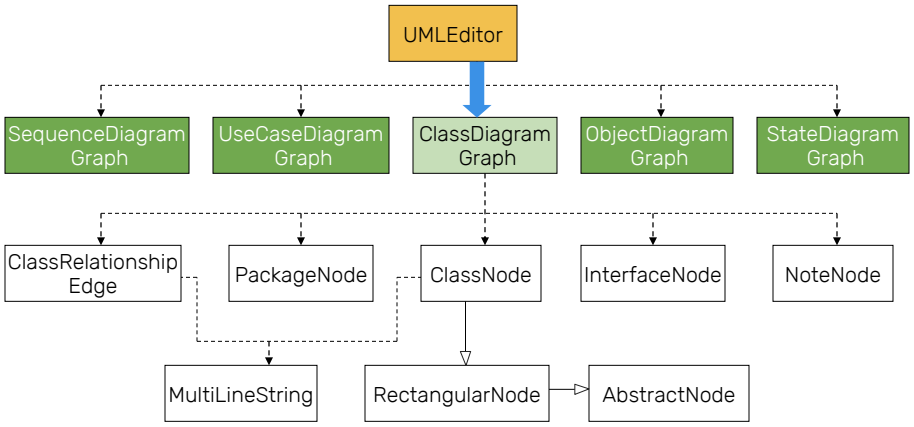
44

Classes to inspect



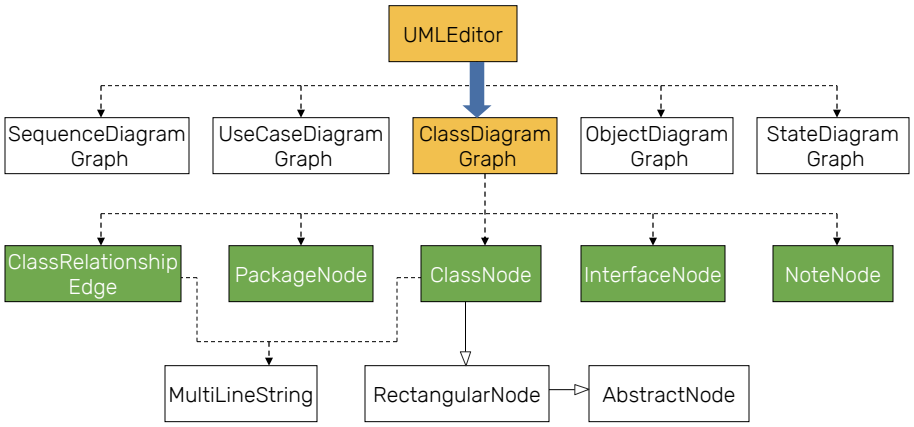
45

Most likely supplier



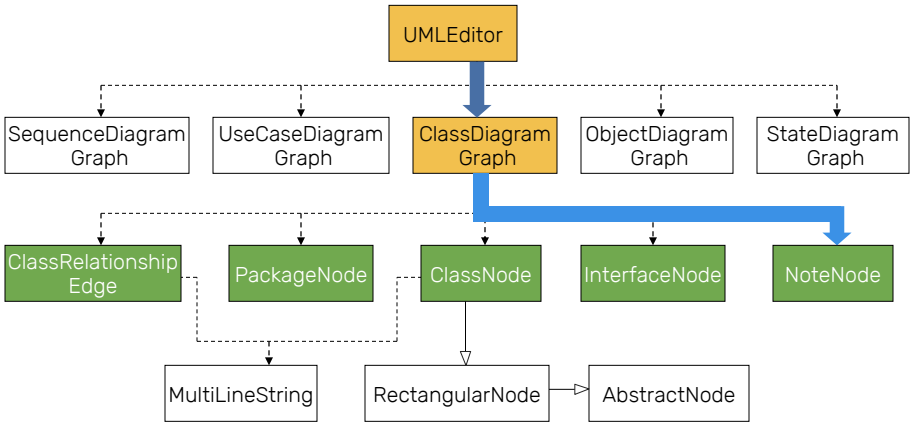
46

Next classes to inspect



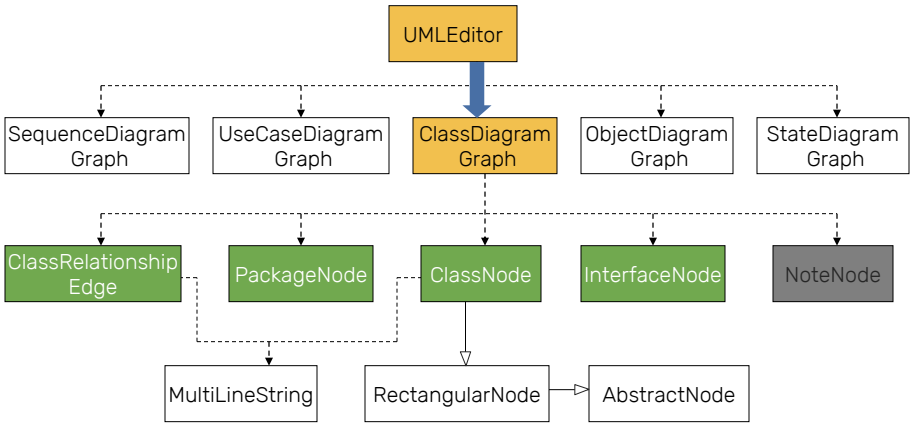
47

Wrong way



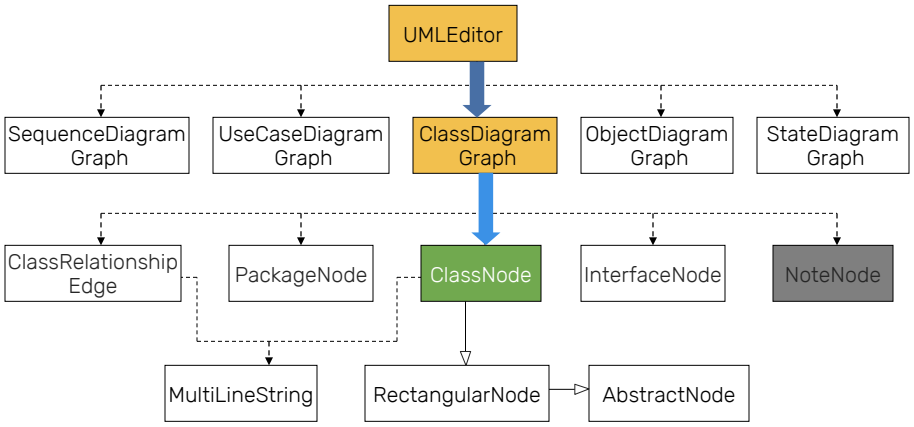
48

Backtrack



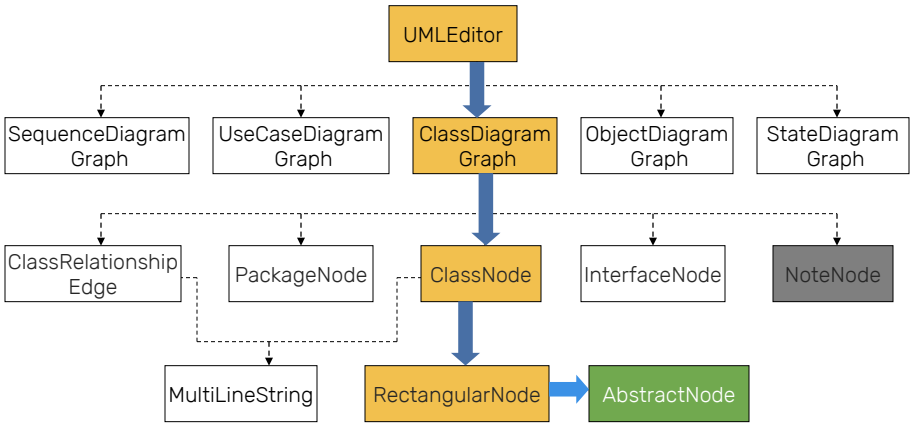
49

Concept found



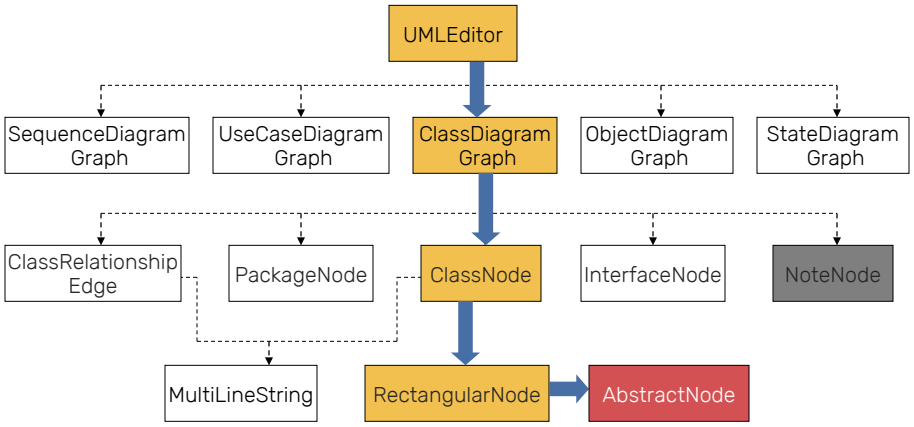
50

Possible extension of the search



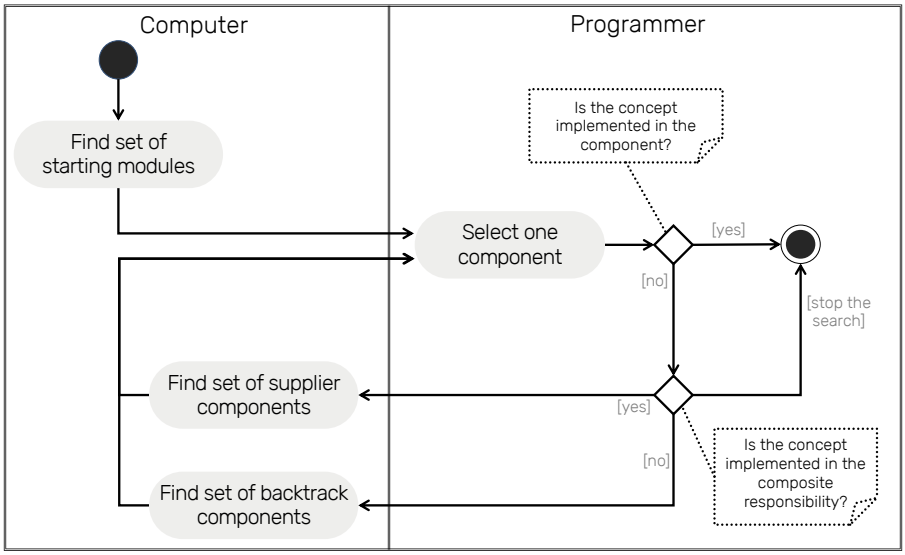
51

Location found



52

Interactive tool for concept location



53

Comparison of the techniques

Text search

- depends on the use of naming conventions
- independent of class structure
- suitable for explicit concepts only

Static dependency search

- utilizes the class structure
- needs correct understanding of composite and local functionality
- suitable for both explicit and implicit concepts

54