Université de Paris

Génie Logiciel
Implementation

Sylvain Lobry

03/12/2021

Resources: www.sylvainlobry.com/GenieLogiciel

Before we start

- Pas de TD4 (vendredi après-midi) cette semaine
- La semaine prochaine, amphi Claude Bernard
- Examen:
  - Pas d'examen blanc
  - Questions de cours (type wooclap/quiz TD)
  - Étude(s) de cas

Software design

Back on design patterns

- 3 patterns seen in class:
  - Singleton (creational pattern)
  - Decorator (structural pattern)
  - Observer (behavioral pattern)
- 20 others!

Software design

Back on design patterns

- Pros:
  - Common solutions, tried and tested
  - Easy to communicate
- Criticisms:
  - High-level programming language often gives solutions to these problems.
  - Often, people try to apply them *too much*. Design patterns are not the solutions to all problems.
  - Specific solution might better fit your project.
  - Use them sparsely
- Discover more: https://refactoring.guru/design-patterns

Table of contents

Programming

How to choose the proper programming language?

- Technical knowledge of the development team
- Technical knowledge of the client
- Interface with other softwares / libraries
- Portability towards different operating systems
- The right level of abstraction
- The right programming paradigm for your project

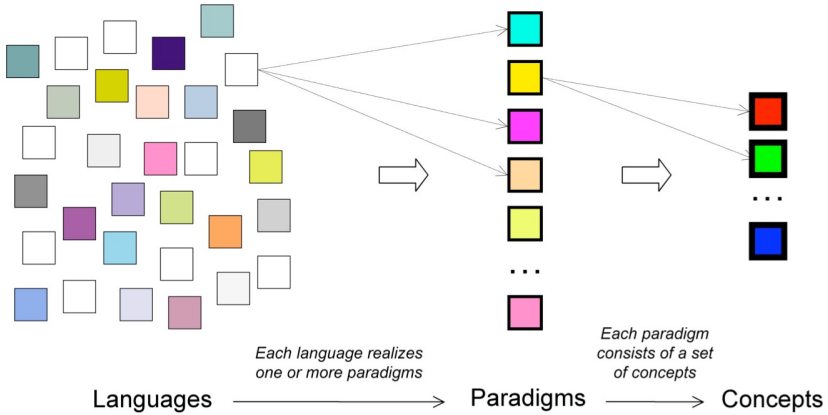
Programming paradigms

Programming paradigms

- Definition:

A programming paradigm *is a style of programming a computer that is defined by a specific set of programming concepts and techniques, as embodied by its* kernel language*, the small core language in which all the paradigm's abstractions can be defined.*

Peter Van Roy

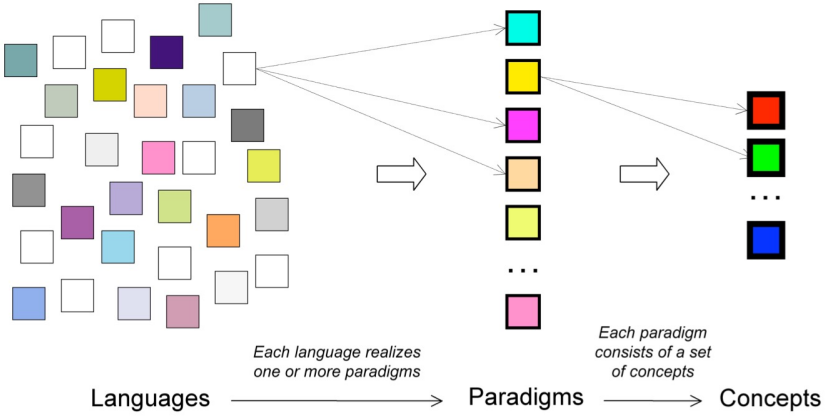


Peter Van Roy, Concepts, Techniques, and Models of Computer Programming, MIT Press

Programming paradigms

Programming paradigms

- A language can realize one paradigm: "*pure*" languages
- Most languages allow several paradigms
- Some languages are designed to realize several paradigms: *multi-paradigm* languages
- A language can evolve and realize new paradigms



Peter Van Roy, Concepts, Techniques, and Models of Computer Programming, MIT Press

Programming paradigms

The ones to know

- Imperative: specify **instructions** to the program (to get to a **result**)

- Declarative: specify **result** to the program (not the **instructions**)

Programming paradigms

The ones to know

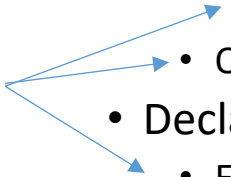
- Imperative: specify **instructions** to the program (to get to a **result**)
  - Structured programming: uses structured control flow (conditions, loops)
    - Procedural programming: uses procedures to structure the program
  - Object-oriented programming: concept of objects (data + code)
- Declarative: specify **result** to the program (not the **instructions**)
  - Functional programming: a program is a composition of functions
  - Logic: expression in terms of logical formulas

Programming paradigms

The ones to know

- Imperative: specify **instructions** to the program (to get to a **result**)
  - Structured programming: uses structured control flow (conditions, loops)
    - Procedural programming: uses procedures to structure the program
  - Object-oriented programming: concept of objects (data + code)
- Declarative: specify **result** to the program (not the **instructions**)
  - Functional programming: a program is a composition of functions
  - Logic: expression in terms of logical formulas

C++

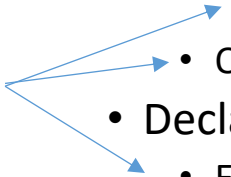


Programming paradigms

The ones to know

- Imperative: specify **instructions** to the program (to get to a **result**)
  - Structured programming: uses structured control flow (conditions, loops)
    - Procedural programming: uses procedures to structure the program
  - Object-oriented programming: concept of objects (data + code)
- Declarative: specify **result** to the program (not the **instructions**)
  - Functional programming: a program is a composition of functions
  - Logic: expression in terms of logical formulas

Python



Programming paradigms

The ones to watch

- Generic: programms are written for generic types (to be specified when used)
- Metaprogramming: program programms
- …. Want to know more? Book chapter from Peter Van Roy

Table of contents

Programming

General rules for clean code

- Need for coding standards:
  - Uniformization of codes written by different developers.
  - Improves readability
  - Improves maintainability
  - (Can) reduce complexity
  - Improves reusability
  - Helps to detect errors
  - Increase efficiency
- Be consistent!
- Example: Google's coding standard for Python

Programming

General rules for clean code

- Give proper names to your variables:
  - variable should reveal its content (e.g. not int i)
  - at least, they should not be misleading (long variable names almost the same, list when it is a vector…)
  - when 2 variables share intent, make meaningful distinction in the name (e.g. not i1, i2, …)
  - You talk to people; try to use names that you can pronounce
  - You should be able to search for your variable (e.g. not i)

Programming

General rules for clean code

- A function should be small
- A function should do one thing
- It should have a descriptive name
- A function should be at a single level of abstraction (no IO with the computation of something for instance)
- It should not have side effect (unexpected from the pov of the caller)
- It should have few arguments:
  - Solution is often to group arguments in semantic objects
- Do not return error codes (use exceptions)

Programming

General rules for clean code

- If the code is clear, do not explain with a comment. Try to fix the code!
- A good comment is either:
  - informative
  - explaining an intent
  - clarifying when no other option
  - Stating a TODO
- A comment should not be:
  - redundant with the code
  - mandatory (it would probably redundant then)
  - dead code (if you see it, delete the code, nobody knows why we need it)

Programming

General rules for clean code

- A good code should be well formated
- Besides obvious indentation (always, even for short if statements or loops)
- Use vertical space to separate concepts
- Functions should have a limited scope
- Functions should be ordered from the most high-level at the top to lower level below (can be problematic for C)

Table of contents

Programming

Test driven development

- 3 laws (from Robert Martin):
  1. You may not write production code until you have written a failing unit test (unit test first)
  2. You may not write more of a unit test than is sufficient to fail, and not compiling is failing (only one failing unit test at a time)
  3. You may not write more production code than is sufficient to pass the currently failing test (the code should only solve the unit test)
- One concept per test
- Repeatable

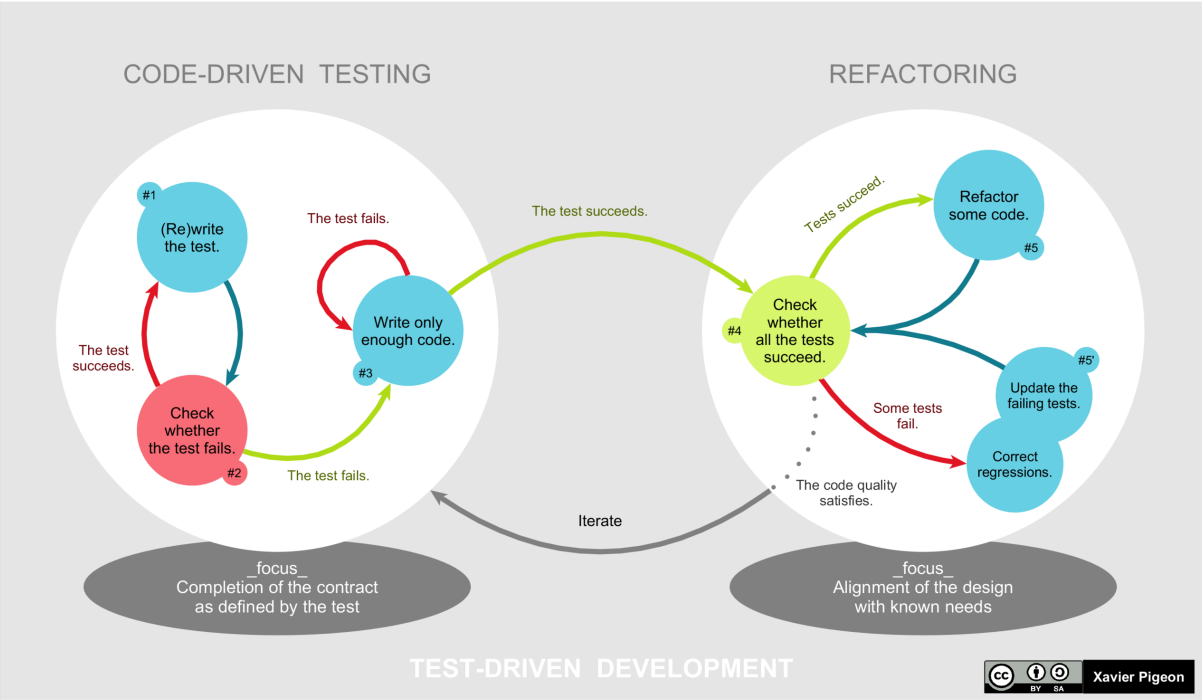
Programming

Test driven development

- 5 stages to add a new feature:
  1. Add a test that will pass **if and only if** the given requirement is met
  2. Run all test. **The new test should fail** as the feature has not been implemented yet.
  3. Add the **minimal amount** of code to pass the test
  4. Check that all tests (including previous ones) are passing
  5. Refactor to **improve readability and maintanability**

Programming

Test driven development



By Xarawn - Own work, CC BY-SA 4.0
https://commons.wikimedia.org/w/index.php?curid=44782343