Universitatea Tehnica din Cluj-Napoca
Departament Calculatoare

OO Programming Techniques in Java

Software Construction

I. Salomie, C. Pop
2017

Objective and Overview

Part 1 – Software Construction
- Problems and Solutions
- Software development process and Software Construction
- Design Heuristics and Best Practices

Part 2 - Programming Paradigms
- Computational Models
- Fundamental Concepts
- Sequential Programming
- Concurrent Programming
- Parallel Programming
- Distributed Programming
- Classes of Algorithms

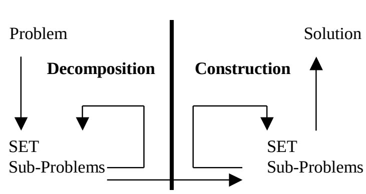
Problems and Solutions

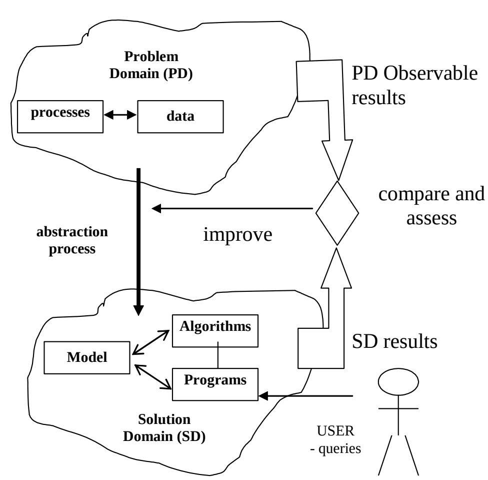
- Problems
- Problem Domain and Solution Domain
- Solution
- Understanding the problem
- Understanding the solution

Problem - Solution

Decomposition / Construction view

- Problem Decomposition
- Solution Construction



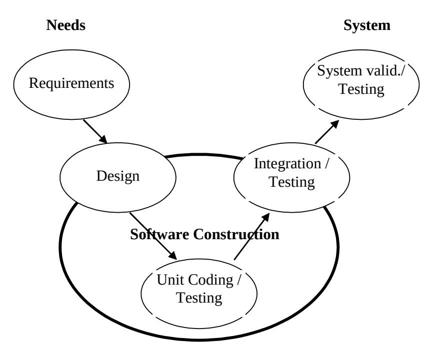


Problem - Solution
Modeling gap

- Modeling Semantic Gap
  - Programs to narrow the gap
- Classical Modeling
- OO Modeling
  - data abstraction and ADTs
  - class dual meaning

Software development process
Software Construction

- Software development process (also called life cycle)
  - input: system requirements
  - output: a delivered product
  - CASE tools to automate some tasks
- Software construction - a three phase process:
  - Detailed design
  - Coding / testing the units
  - Solution composition



Software Construction
Details

- Design Level 1 - Overall system design
  - The big black box
- **Design Levels 2,3,4 – Detailed Design**
  - **Design Level 2 - Subsystem/package level design**
    - **for each subsystem: Interfaces, Classes, Relationships**
  - **Design Level 3 - Class level design**
  - **Design Level 4 - Data and function level design**
- **Unit coding / Unit testing**
- **Integration planning / testing (Subsystem level)**
- System integration and testing planning

Design

- Objective: Obtain a good (sub)system
  - (partial) reuse existent resources;
  - (partial) new resources
- Design
  - iterative try-error and divide and conquer type activity
- Breaking complexity
  - top-down or bottom-up approaches
- Skilled designers
  - Use heuristics
- Desirable Design Features
  - External Features
  - Internal Features

Desirable Design Features

External and Internal Design Features

External Design Features

- Availability
- Reliability
- Performance
- Fault tolerant
- Maintainability
- Robustness
- Security
- Interoperability
- Portability
- Usability
- Functionality
- System integrity
- Consistency
- Efficiency

Internal Design Features

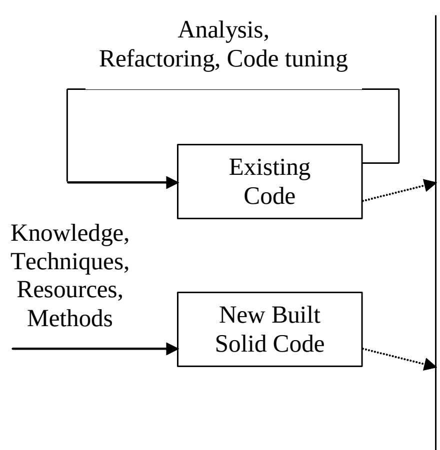
- Minimal complexity
- Ease of maintenance
- Loose coupling among design (program) parts
- Extensibility / Scalability
- Reusability
- Stratification
- Standard techniques

Quality Models

- Relates external criteria to external criteria
- Models
  - McCall (1977)
  - Boehm (1978)
  - ISO 9126 (1990)
  - Dromey (1996)

Software construction

Objectives and Features



Features

External
Reliability
Performance
Failure tolerant
Maintenance
Portability
Interoperability
Security
Testability

Internal
Reusable
Extensible / Scalable
Low complexity
Loose coupling
High fan-in
Low to medium fan-out
Completeness
Standardized

Software Construction Heuristics

- Heuristic [Wikipedia]
  - A replicable method or approach for directing one's attention in learning, discovery, or problem-solving
- George Polya - ‘How to solve it’
  - Collection of ideas about heuristics
  - Examples from ‘How to solve it’
    - If you are having difficulty understanding a problem, try drawing a picture
    - If you can't find a solution, try assuming that you have a solution and seeing what you can derive from that ("working backward")
    - If the problem is abstract, try examining a concrete example
    - Try solving a more general problem first (the "inventor's paradox": the more ambitious plan may have more chances of success)
- Nature and bio-inspired heuristics

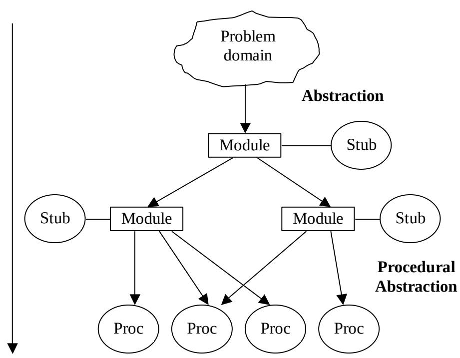
Software Construction
Main Heuristics

- Find real-world objects
- Build abstractions and hierarchies of abstractions
- Encapsulate implementation details
- Inherit or delegate
- Identify areas likely to change
- Keep coupling loose and coherence high
- Look for common design patterns
- Think of associating things
- Design for test
- Choose carefully binding time
- Draw diagrams
- Keep the design modular

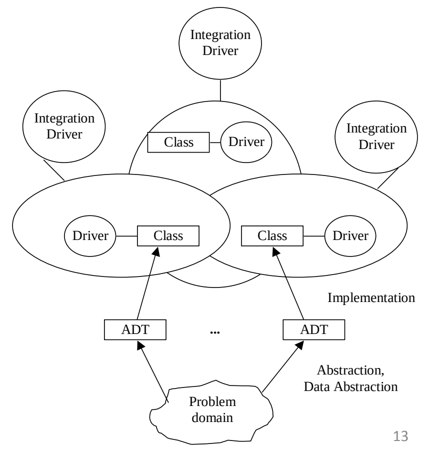
Design Practices

Divide et Impera - Top Down (Procedural Development)

Divide et Impera - Top Down (Procedural Development)



Divide et Impera – Bottom Up (OO development))



Part 2 Programming Paradigms

Objective

- Presenting the main computational models and programming paradigms

Programming features

- (Computer) Programming
  – science of creating software for solving problems
  – both knowledge intensive and
  – labor intensive process
- Problem dimensions
  – simple, complex,
  – small, large,
  – IO intensive,
  – algorithmic intensive,
  – data intensive
- Modern software development tools
  – complex tools embedding
    - software construction knowledge,
    - reuse at different levels

Computational Models

Computational Model	Main Abstractions
Procedural Oriented (Imperative Programming)	Procedures, Algorithms Substitution principle Ex: Pascal, C, (partial) C++
Object Oriented	Objects and classes Ex: Simula, SmallTalk, C++, Java, C#
Functional	Functions, Lambda Calculus Ex: Lisp. ML, Haskel
Logic	Goals (uses predicate calculus) Ex: Prolog, SQL
Constraint Oriented	Invariant relationships

Choosing the right programming language
- Difficult problem
- Problem - Programming Language mapping
- Are there any decision metrics??

Classes of Algorithms and Problems

Type	Uniprocessor	Multiprocessor
Sequential	Sequential Execution	Multi-programming
Concurrent	Apparent concurrency	Real Concurrency

Sequential and Concurrent Programming

Sequential Programming
- Execution of one statement at a time
- Usually executed on a single processor
  - Processor speed is important
- Transforms input data to results according to the implemented algorithm
- Usually deterministic

Concurrent Programming
- Concurrent program
  - Programming entities - executed simultaneously (in parallel)
  - Dynamically information sharing
- Large variety of concurrent models
  - Apparent or real concurrency
- Execution support
  - mono, multi, **parallel** or **distributed** processes
- Transform sequential to concurrent programming
  - Strategy to decompose large tasks into multiple smaller tasks (that execute concurrently)
  - Assigning the smaller tasks to multiple workers to work on simultaneously.
  - Coordinating the workers

Sequential and Concurrent Programming

Parallel Programming

- Special form of concurrent programming
  - Multiple physical processors
  - Memory sharing
- Approaches to constructing parallel applications:
  - Functional parallelism
  - Master–slave parallelism
  - SPMD parallelism (same code, replicated to each process)
- Parallel programming main tasks
  - Decomposing tasks
  - Distributing tasks
  - Coordinating tasks

Distributed Programming

- Special form of concurrent programming
  - multiple physical processors
  - remotely located
  - no shared memory
- Interprocess communication
  - communication channels
  - message exchange

Transformational and Reactive Programming

Transformational Programming

- Based on the sequence:
  - read data
  - process data
  - output results
- Can be seen as functions of n inputs generating m outputs
- Program behavior depends on program current state and input
- Can be parallelized

Reactive Programming

- Interact with the environment during execution;
- Cannot be specified as a function
- Event driven programming
- Inherently non-deterministic