Development Methodologies

Prof. Dr. Josef M. Joller
jjoller@hsr.ch

THE TOOLS OF THE TRADE

Overview

Stepwise refinement

Cost–benefit analysis

Software metrics

CASE

Taxonomy of CASE

Scope of CASE

Software versions

Configuration control

Build tools

Stepwise Refinement

A basic principle underlying many software engineering techniques

- “Postpone decisions as to details as late as possible to be able to concentrate on the important issues”

Miller’s law (1956)

- A human being can concentrate on 7±2 items at a time

Appraisal of Stepwise Refinement

A basic principle used in

- Every phase
- Every representation

The power of stepwise refinement

- The software engineer can concentrate on the relevant aspects

Warning

- Miller’s Law is a fundamental restriction on the mental powers of human beings

Cost–Benefit Analysis

Compare estimated future benefits, costs

- Estimate costs
- Estimate benefits
- State all assumptions explicitly

Sources

- Boehm : Software Engineering Economics (old but still okay)
- Inhouse project data

CASE (Computer-Aided Software Engineering)

Scope of CASE

- Can support the entire life-cycle

Graphical display tools (many for PCs)

- Data flow diagrams
- Entity-relationship diagrams
- Module-interconnection diagrams
- Petri nets
- Structure charts

Software Metrics

To detect problems early, it is essential to measure

Examples:

- LOC per month
- Defects per 1000 lines of code
- Number of screens
- Number of reports
- Number of objects
- …

Different Types of Metrics

Product Metrics

- Examples:
  - Size of product
  - Reliability of product

Process Metrics

- Example:
  - Efficiency of fault detection during development

Metrics specific to a given phase

- Example:
  - Number of defects detected per hour in specification reviews

The Five Basic Metrics

Size

- In Lines of Code, or better

Cost

- In dollars

Duration

- In months

Effort

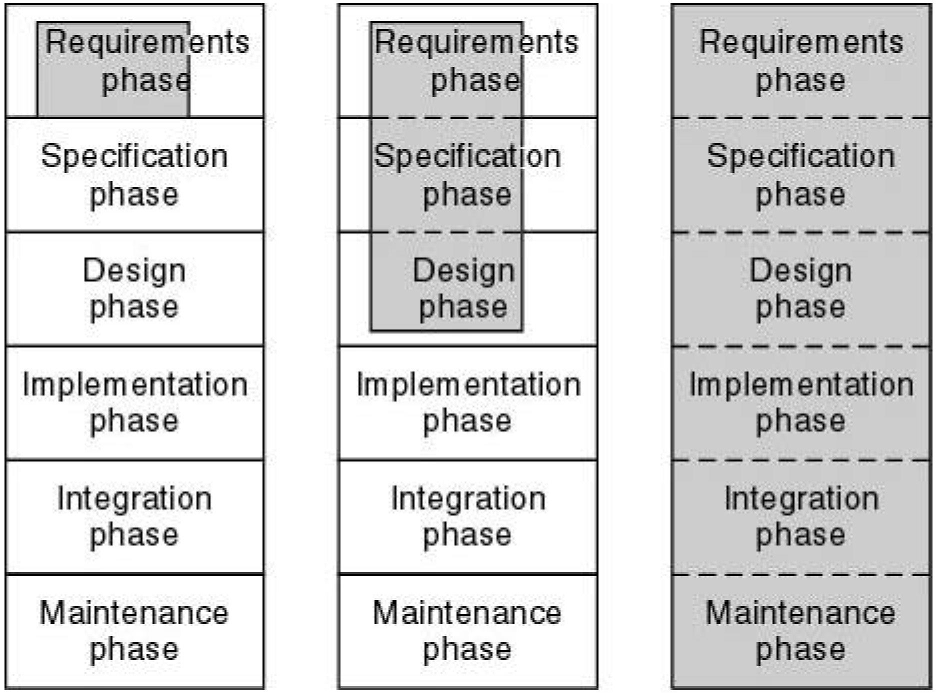
- In person months

Quality

- Number of faults detected

Taxonomy of CASE

UpperCASE versus lowerCASE



(a) Tool (b) Workbench (c) Environment

Tool versus workbench versus environment

Graphical Tool

Additional features

- Data dictionary
- Screen and report generators
- Consistency checker; the various views are always consistent
  - Specifications and design *workbench*

Online Documentation

- Problems with
  - Manuals
  - Updating

Essential online documentation

- Help information
- Programming standards
- Manuals

Essential Coding Tools

Coding tools

- Products (such as text editors, debuggers, and pretty printers, interface checkers) designed to
  - Simplify programmer's task
  - Reduce frustration
  - Increase programmer productivity

Conventional coding scenario for *programming-in-the-small*

- Editor-compiler cycle
- Editor-compiler-linker cycle
- Editor-compiler-linker-execute cycle

"There must be a better way"

Syntax-directed Editor

“Understands” language

- Speeds up implementation
- User interface of an editor is different to that of a compiler
  - There is no need to change thinking mode
  - No mental energy is wasted on these adjustments
- One piece of system software, two languages
  - High-level language of module
  - Editing command language
- Pretty-printer

Source Level Debugger

The programmer works in a high-level language, but must examine

- Machine code core dumps
- Assembler listings
- Linker listings
- Similar low-level documentation

Destroys the advantage of programming in a high-level language

We need

- Interactive source level debugger

Programming Workbench

Structure editor with

- Online interface checking capabilities
- Operating system front-end
- Online documentation
- Source level debugger

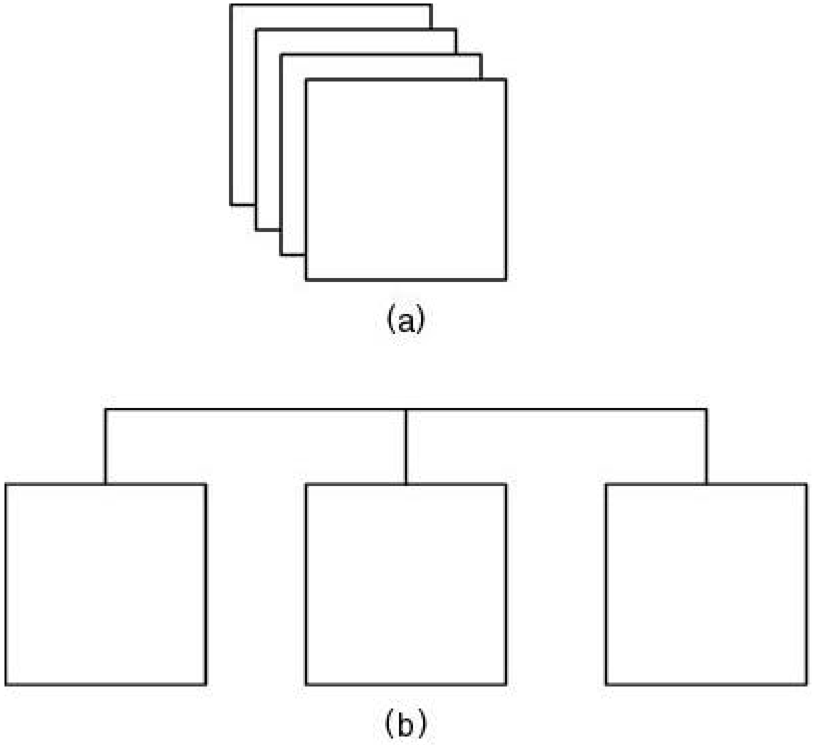
Constitutes a simple programming environment

This is by no means new

- All the above features are supported by FLOW (1980)
- The technology has been in place for years

Surprisingly, some programmers still implement code Ye Olde-Fashioned Way

Revisions and Variations



Variation

- Version for different operating system–hardware
- Variations are designed to coexist in parallel

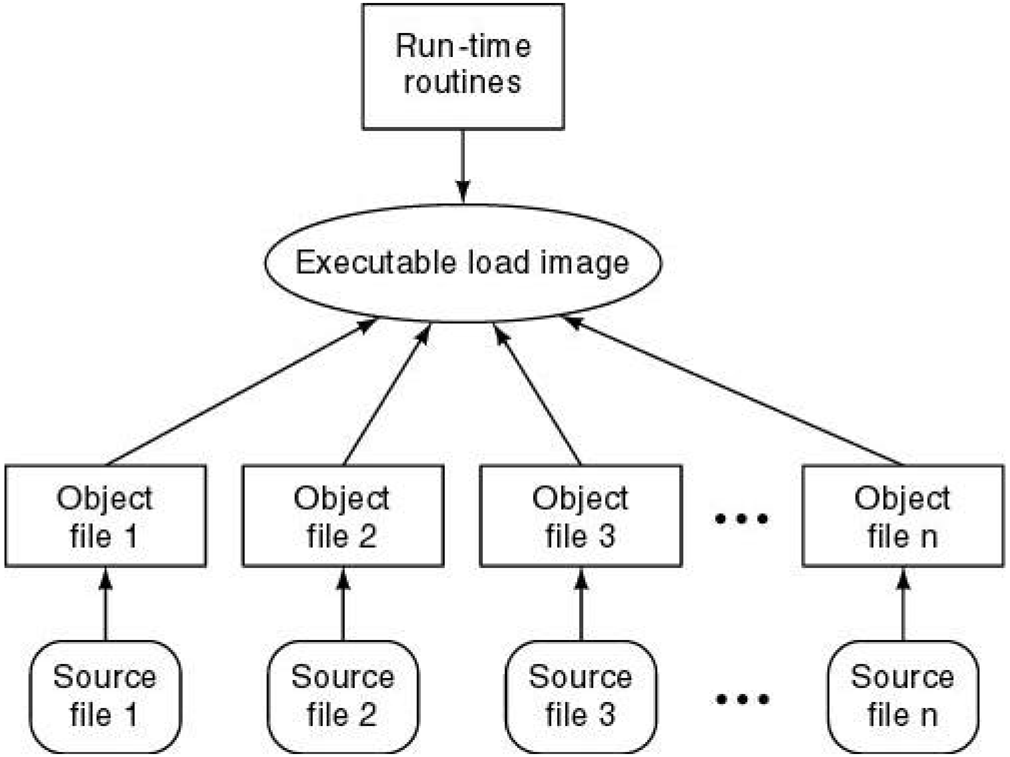
Configuration Control

Every module exists in three forms

- Source code; object code; executable load image

Configuration

- Version of each module from which a given version of a product is built



Build Tools

Example

- UNIX *make*
- Apache *ant*

Compares the date and time stamp on

- Source code, object code
- Object code, executable load image

Can check dependencies

- Ensures that correct versions/variations are compiled and linked

Productivity Gains with CASE Tools

Survey of 45 companies in 10 industries [Myers, 1992]

- Half information systems
- Quarter scientific
- Quarter real-time aerospace

Results

- About 10% annual productivity gains
- $125,000 per seat

Justifications for CASE

- Faster development
- Fewer faults
- Easier maintenance
- Improved morale