Stepwise Refinement

- Stepwise refinement is a basic problem-solving principle underlying many software engineering techniques. It can be defined as:
  - A means to 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 approximately 7 (i.e., 7 ± 2) items at a time

Stepwise Refinement Mini Case

- Design a product to update a sequential master file containing name and address data for the monthly magazine *True Life Software Disasters*
- **Three types of transactions**
  - Type 1: INSERT (a new subscriber into the master file)
  - Type 2: MODIFY (an existing subscriber record)
  - Type 3: DELETE (an existing subscriber record)
- Transactions are sorted into alphabetical order by name of subscriber, and by transaction code within alphabetical order

Mini Case: Decompose Process

- Typical Input Transaction File

Mini Case: The First Refinement

- Assumption
  - We can produce a record when PROCESS requires it
- Solution
  - Separate INPUT and OUTPUT, concentrate on PROCESS

Mini Case: What is this PROCESS?

- Example:

<table>
<thead>
<tr>
<th>Transaction file</th>
<th>Old master file</th>
<th>New master file</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Brown</td>
<td>Abel</td>
<td>Abel</td>
</tr>
<tr>
<td>1 Harris</td>
<td>Brown</td>
<td>Harris</td>
</tr>
<tr>
<td>2 Jones</td>
<td>James</td>
<td>James</td>
</tr>
<tr>
<td>3 Jones</td>
<td>Jones</td>
<td>Smith</td>
</tr>
<tr>
<td>1 Smith</td>
<td>Smith</td>
<td>Townsend</td>
</tr>
</tbody>
</table>

  Exception report
  - Smith
Mini Case: The Second Refinement

- This design has a major fault since: “Modify JONES” followed by “Delete JONES” is incorrectly handled.

Mini Case: The Third Refinement

- After the third refinement has been corrected, details like opening and closing files have to be introduced. Fix these after the logic of the design is complete.
- The important issue is to decide which item should be handled at each stage!

Appraisal of Stepwise Refinement

- It can be considered a technique for setting the priorities of the various problems that have to be solved within a workflow.
- It ensures that every problem is solved and each is solved at the appropriate time, without having to handle more than 7 ± 2 chunks at any one time.
- It is a basic and general principle used in every workflow and every representation (e.g., flowchart and pseudocode).
- The power of stepwise refinement: The software engineer can concentrate on the relevant aspects.

Cost–Benefit Analysis

- Compare costs and future benefits:
  - Estimate costs
  - Estimate benefits
  - State all assumptions explicitly

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salary savings (7 years)</td>
<td>$1,575,000</td>
</tr>
<tr>
<td>Improved cash flow (7 years)</td>
<td>$875,000</td>
</tr>
<tr>
<td>Total benefits</td>
<td>$2,450,000</td>
</tr>
<tr>
<td>Hardware and software (7 years)</td>
<td>$1,250,000</td>
</tr>
<tr>
<td>Conversion cost (first year only)</td>
<td>$350,000</td>
</tr>
<tr>
<td>Explanations to customers (first year only)</td>
<td>$125,000</td>
</tr>
<tr>
<td>Total costs</td>
<td>$1,725,000</td>
</tr>
</tbody>
</table>

Cost–Benefit Analysis (Cont.)

- Tangible costs/benefits are easy to measure.
- Make assumptions to estimate intangible costs/benefits:
  - Improving the assumptions will improve the estimates.
Software Metrics (Measurements)

- To detect problems early in the software process, it is essential to measure
  - LOC (Lines of Code) per month
  - Defects per 1000 lines of code
- Once the product has been installed on the client’s computer, it essential to have **product metrics**
  - Size of product
  - Mean time between failures: Measure the reliability of product
- It is important to have **process metrics** to help developers to deduce information about the software process.
  - Efficiency of fault detection during development
- Metrics specific to a given workflow
  - 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

CASE (Computer-Aided Software Engineering)

- Scope of CASE
  - CASE can support the entire life-cycle
- The computer assists with drudge work
  - It manages all the details including plans, contracts, specifications, designs, source code, and management information

Scope of CASE

- Programmers need to have:
  - Accurate, up-to-date versions of all documents
  - Online help information regarding the operating system, editor, and programming language.
  - Online programming standards
  - Online manuals including editor manuals and programming manuals
  - Online interface checkers
  - E-mail systems
  - Spreadsheets, word processors
  - Structure editors
  - Pretty printers

CASE Tool: Online Interface Checker

- A structure editor must support online interface checking
  - The editor must know the name of every procedure
- Interface checking is an important part of programming-in-the-large (software development at the module level)

CASE Tool: Online Interface Checker - Example 1

- The user enters the call:
  \[ \text{Average} = \text{dataArray.computeAverage(Values)} \]
- The editor immediately responds
  - Method computeAverage not known
  - This enables full interface checking
  - Correct the method name to computeMean
  - Declare new procedure computeAverage and specify its parameters
CASE Tool: Online Interface Checker - Example 2
• Declaration of method q is
  ```
  void q(float floatVar, int intVar, String s1, String s2);
  ```
• Call (invocation) is
  ```
  q(intVar, floatVar, s1, s2);
  ```
• The online interface checker detects the fault
• Help facility
  – Online information for the parameters of q
  – Better: Editor generates a template for the call
    • The template shows the type of each parameter
    • The programmer replaces formal by actual parameters

CASE Tool: Online Interface Checker - Summary
• Advantages
  – Hard-to-detect faults are immediately flagged for correction
  • Wrong number of parameters
  • Parameters of the wrong type
  – Essential when software is produced by a team
    • If one programmer changes an interface specification, all components calling that changed artifact must be disabled

CASE Tool: Operating System Front-End within the Editor
• Single command
  – go or run
  – Use of the mouse to choose
    • An appropriate icon, or
    • An appropriate menu selection
• This one command causes the editor to invoke the compiler, linker, loader, and execute the product

CASE Tool: Source Level Debugger
• An interactive source level debugger (like dbx) can generate output from a typical source level

Other Useful CASE Tools
• Data dictionary: A computerized list of all data defined within the product
• Consistency checker: A tool to check that every data item in the specification document is reflected in the design, and conversely, every item in the design has been defined in the specification document.
• Report generator: Generate the code needed for producing a report.
• Screen generator: Assist the software developer in producing the code for a data capture screen.

Programming Workbench
• Structure editor with
  – Online interface checking capabilities
  – Operating system front-end
  – Online documentation
  – Source level debugger
• This constitutes a simple, adequate, and effective programming environment
Taxonomy of CASE

- **UpperCASE (front-end tool):** Help the developer during the earlier workflows of the process (the requirements, analysis, and design workflows)

- **LowerCASE (back-end tool):** Assist with the implementation workflow and postdelivery maintenance.

---

Software Versions

- During maintenance, at all times there are at least two versions of the product:
  - The old version, and
  - The new version

- There are two types of versions: *revisions* and *variations*

---

Variations

- A variation is a version for different operating systems – hardware

- Variations are designed to coexist in parallel

---

Revisions

- **Revision**
  - A version to fix a fault in the artifact (Corrective Maintenance)
  - We cannot throw away an incorrect version
  - The new version may be no better
  - Some sites may not install the new version

- Perfective and adaptive maintenance also result in revisions

---

Configuration Control

- Every code artifact exists in three forms
  - Source code
  - Object code
  - Executable load image

- **Configuration**
  - A version of each artifact from which a given version of a complete product is built
Version-Control Tool

- Essential for programming-in-the-many (software production by a team)
  - A first step toward configuration management

- A version-control tool must handle
  - Updates
  - Parallel versions

Examples of Notation for file name, variation, and version (see the left diagram)

Version-Control Tool: Advantages and Problems

- Version control is a great help in managing multiple versions of artifacts and the product as a whole.

- However, additional problem associated with maintaining multiple variations
  - The fault occurs in a part of the artifact common to both variations
  - Solution: The variation is stored in terms of the list of changes that have to be made to go from the original to that variation. That is, save the Deltas!

- Version control is not enough — maintenance issues

Configuration Control During Postdelivery Maintenance: Problems

- Two programmers are working on the same artifact mDual/16

- The changes of the first programmer are contained in mDual/17

- The changes of the second programmer are contained in mDual/18
  - The changes of the first programmer are lost

Solution: Baselines

- The maintenance manager must set up
  - Baselines: A configuration (set of versions) of all the artifacts in the product
  - Private workspaces

- When an artifact is to be changed, the current version is frozen
  - Thereafter, it can never be changed

- Both programmers make their changes to mDual/16
  - The first programmer
    - Freezes mDual/16 and makes changes to it
    - The resulting revision is mDual/17
    - After testing, mDual/17 becomes the new baseline
  - The second programmer
    - Freezes mDual/17 and makes changes to it
    - The resulting revision is mDual/18
    - After testing, mDual/18 becomes the new baseline

Configuration Control during Development

- While an artifact is being coded
  - The programmer performs informal testing

- Then the artifact is given to the SQA group for methodical testing
  - Changes from now on can impact the product

- An artifact must be subject to configuration control from the time it is passed by SQA

Configuration-Control Tools: Examples

- UNIX version-control tools
  - sccs
  - rcs
  - cvs

- Popular commercial configuration-control tools
  - PVCS
  - SourceSafe

- Open-source configuration-control tool
  - cvs
Build Tools

- A build tool assists in selecting the correct version of each compiled-code artifact to be linked to form a specific version of the product. Example: UNIX make

- It compares the date and time stamp on
  - Source code, object code
  - It calls the appropriate compiler only if necessary

- It then compares the date and time stamp on
  - Object code, executable load image
  - It calls the linker only if necessary

Productivity Gains with CASE Tools

- Introducing CASE technology into a software organization leads to:
  - Faster development
  - Fewer faults
  - Better usability
  - Easier maintenance
  - Improved morale

- Results confirm that CASE environments should be used at CMM level 3 or higher