Chapter 10

Users of a System

- Users
  - Exercise main system functions
  - Use the system to solve customer’s problems
- Operator
  - Perform supplemental functions
### Table 10.1. User and operator functions.

<table>
<thead>
<tr>
<th>User functions</th>
<th>Operator functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manipulating data files</td>
<td>Granting user access</td>
</tr>
<tr>
<td>Simulating activities</td>
<td>Granting file access</td>
</tr>
<tr>
<td>Analyzing data</td>
<td>Performing backups</td>
</tr>
<tr>
<td>Communicating data</td>
<td>Installing new devices</td>
</tr>
<tr>
<td>Drawing graphs and charts</td>
<td>Installing new software</td>
</tr>
<tr>
<td></td>
<td>Recovering damaged files</td>
</tr>
</tbody>
</table>

#### User Training

- **Major System Functions**
- Need not be aware of system internal operation.
- Relate to old system
  - Psychological barrier to learning new system
Operator Training

- Support Functions
- How the system works, not what it does
  - Start-up/Shut-down
  - Configuration
  - Grant/Deny access
  - Assign resources (e.g., disk space)
  - Monitor & tune system performance
  - Recover lost files

Special Training Needs

- Common overview for both users and operators.
- Individualized training for new users/operators.
- Refresher training for infrequently used functions.
Training Aids

- Documentation
- Icons
- On-Line Help
- Demonstrations
- Classes
- Expert Users

Guides for Training

- Training is successful only when it meets your needs and matches your capabilities.
  - Some students learn by reading, others by hearing, and others by using a combination.
- Individualized systems can accommodate variation in backgrounds, experience, and preferences.
- Material should be divided into units of limited scope.
  - Too much material at once can be overwhelming.
Documentation

- User manual
- Operator manual
- General system guide
- Tutorials and automated overviews
- Programmer guide

User’s Manual

- The system’s purpose and objectives
- The system’s capabilities and functions
- The system’s features, and advantages
Operator’s Manual

• Hardware and software configuration
• Methods for granting/denying access
• Adding or moving peripherals
• Duplicating or backing up data files

Chapter 10

General System Guide

• Overview of what the system does, without details.
• Hardware and software configuration.
• Cross reference to other documentation to provide details.

Chapter 10
Tutorials

• Some users prefer to be guided through actual system functions.
• Documentation combined with special program to guide the user through the functions step-by-step.

Programmer Guide

• Detailed description of the software architecture.
• Guide to the maintenance staff.
• Explain tools available for updating the code and corresponding documentation.
Guidelines for failure messages

- the name of code component executing when the failure occurred
- the source code line number in the component that was executing
- the failure severity and its impact on the system
- the contents of any relevant system memory or data pointers, such as registers or stack pointers
- the nature of the failure, or a failure message number (for cross-reference with the failure message reference guide)

Example failure messages

> 3 x
Syntax error: name ("x") used illegally at this point:
3 x

> .5(2,4)
Error: "0.5" is not a function

FAILURE 345A1: STACK OVERFLOW
Occurred in: COMPONENT DEFRECD
At line: 12300
Severity: WARNING
Register contents: 0000 0000 1100 1010 1100 1010 1111 0000
Press function key 12 to continue

Failure 345A1: Stack overflow.
This problem occurs when more fields are defined for a record than the system can accommodate. The last field defined will not be included in the record. You can change the record size using the Record Maintenance function on the Maintenance menu to prevent this failure in the future.
Lehman’s system types

- S-system: formally defined, derivable from a specification
- P-system: requirements based on approximate solution to a problem, but real-world remains stable
- E-system: embedded in the real world and changes as the world does
Chapter 10

A P-system

- Real world
- Problem
- Abstraction
- Requirements specification
- System
- Information

Comparison

Subject to change

Chapter 10

An E-system

- Real world
- Problem
- Abstraction
- Requirements specification
- System
- Information

Comparison

Chapter 10
Table 11.1. Examples of change during software development.

<table>
<thead>
<tr>
<th>Activity from which initial change results</th>
<th>Artifacts requiring consequent change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements analysis</td>
<td>Requirements specification</td>
</tr>
<tr>
<td>System design</td>
<td>Architectural design specification</td>
</tr>
<tr>
<td>Program design</td>
<td>Technical design specification</td>
</tr>
<tr>
<td>Program implementation</td>
<td>Program design specification</td>
</tr>
<tr>
<td></td>
<td>Program code</td>
</tr>
<tr>
<td></td>
<td>Program documentation</td>
</tr>
<tr>
<td>Unit testing</td>
<td>Test plans</td>
</tr>
<tr>
<td></td>
<td>Test scripts</td>
</tr>
<tr>
<td>System testing</td>
<td>Test plans</td>
</tr>
<tr>
<td></td>
<td>Test scripts</td>
</tr>
<tr>
<td>System delivery</td>
<td>User documentation</td>
</tr>
<tr>
<td></td>
<td>Training aids</td>
</tr>
<tr>
<td></td>
<td>Operator documentation</td>
</tr>
<tr>
<td></td>
<td>System guide</td>
</tr>
<tr>
<td></td>
<td>Programmer guide</td>
</tr>
<tr>
<td></td>
<td>Training classes</td>
</tr>
</tbody>
</table>

System evolution vs. decline

- Is the cost of maintenance too high?
- Is the system reliability unacceptable?
- Can the system no longer adapt to further change, and within a reasonable amount of time?
- Is system performance still beyond prescribed constraints?
- Are system functions of limited usefulness?
- Can other systems do the same job better, faster or cheaper?
- Is the cost of maintaining the hardware great enough to justify replacing it with cheaper, newer hardware?
Laws of software evolution

• Continuing change: leads to less utility
• Increasing complexity: structure deteriorates
• Fundamental law of program evolution: program obeys statistically-determined trends and has invariants
• Conservation of organizational stability: global activity rate is invariant
• Conservation of familiarity: release content is invariant

Types of maintenance

• Corrective: maintaining control over day-to-day functions
• Adaptive: maintaining control over system modifications
• Perfective: perfecting existing functions
• Preventive: preventing system performance from degrading to unacceptable levels
Chapter 10

Distribution of maintenance

- Perfective 50%
- Adaptive 25%
- Corrective 21%
- Preventive 4%

Chapter 10

Maintenance team responsibilities

- understanding the system
- locating information in system documentation
- keeping system documentation up-to-date
- extending existing functions to accommodate new or changing requirements
- adding new functions to the system
- finding the source of system failures or problems
- locating and correcting faults
- answering questions about the way the system works
- restructuring design and code components
- rewriting design and code components
- deleting design and code components that are no longer useful
- managing changes to the system as they are made
Maintenance problems

• Staff problems
  – Limited understanding
  – Management priorities
  – Morale
• Technical problems
  – Artifacts and paradigms
  – Testing difficulties

Factors affecting maintenance effort

• Application type
• System novelty
• Turnover and maintenance staff ability
• System life span
• Dependence on a changing environment
• Hardware characteristics
• Design quality
• Code quality
• Documentation quality
• Testing quality
Measuring maintainability

• Necessary data:
  – time at which problem is reported
  – time lost due to administrative delay
  – time required to analyze problem
  – time required to specify which changes are to be made
  – time needed to make the change
  – time needed to test the change

  – time needed to document the change

• Desirable data:
  – ratio of total change implementation time to total number of changes implemented
  – number of unresolved problems
  – time spent on unresolved problems
  – percentage of changes that introduce new faults
  – number of components modified to implement a change

Mean Time to Repair

Mean time to repair fault (hours)

System area
Example for calculating cyclomatic number

```cpp
Scoreboard::drawscore(int n)
{
    while(numdigits-- > 0) {
        score[numdigits]->erase();
    }
    // build new score in loop, each time update position
    numdigits = 0;
    // if score is 0, just display "0"
    if (n == 0) {
        delete score[numdigits];
        score[numdigits] = new Displayable(digits[0]);
        score[numdigits]->move(Point((700-numdigits*18),40));
        score[numdigits]->draw();
        numdigits++;
    }
    while (n) {
        int rem = n % 10;
        delete score[numdigits];
        score[numdigits] = new Displayable(digits[rem]);
        score[numdigits]->move(Point(700-numdigits*18),40));
        score[numdigits]->draw();
        n /= 10;
        numdigits++;
    }
}
```

Chapter 10
Fog index

\[ F = 0.4 \times \frac{\text{number of words}}{\text{number of sentences}} + \text{percentage of words of 3 or more syllables} \]

Configuration control process

- Problem discovered by or change requested by user/customer/developer, and recorded
- Change reported to the configuration control board
- CCB discusses problem: determines nature of change, who should pay
- CCB discusses source of problem, scope of change, time to fix; they assign severity/priority and analyst to fix
- Analyst makes change on test copy
- Analyst works with librarian to control installation of change
- Analyst files change report
Change control issues

- **Synchronization**: When was the change made?
- **Identification**: Who made the change?
- **Naming**: What components of the system were changed?
- **Authentication**: Was the change made correctly?
- **Authorization**: Who authorized that the change be made?
- **Routing**: Who was notified of the change?
- **Cancellation**: Who can cancel the request for change?
- **Delegation**: Who is responsible for the change?
- **Valuation**: What is the priority of the change?

Impact analysis

- **Workproduct**: any development artifact whose change is significant
- **Horizontal traceability**: relationships of components across collections of workproducts
- **Vertical traceability**: relationships among parts of a workproduct
Chapter 10

- Manage software maintenance
- Change Request
- Analyze software change impact
- Understand software under change
- Implement maintenance change
- Account for ripple effect
- (Re)test affected software

Preventive
Adaptive
Corrective
Perfuctive

NEW SYSTEM

Impact/scope
Traceability roadmap
Complexity
Modularity
Documentation
Self-descriptiveness
Adaptability
Stability
Consistency
Testability
Verifiability
Completeness

Existing system

Requirements document
r1
r2
r2.2
r3
...

Design components

Code components

Tests

Acceptance test n.2

Test t.1
Test t.2
Test t.3
Test t.4
Test t.5
Test t.6
Test t.7
Test t.8
Test t.9
Test t.10
Test t.11
Test t.12

Chapter 10
Automated maintenance tools

- Text editors
- File comparators
- Compilers and linkers
- Debugging tools
- Cross-reference generators
- Static code analyzers
- Configuration management repositories
Software rejuvenation

- Redocumentation: static analysis adds more information
- Restructuring: transform to improve code structure
- Reverse engineering: recreate design and specification information from the code
- Reengineering: reverse engineer and then make changes to specification and design to complete the logical model; then generate new system from revised specification and design
Redocumentation

- Output may include:
  - component calling relationships
  - data-interface tables
  - data-dictionary information
  - data flow tables or diagrams
  - control flow tables or diagrams
  - pseudocode
  - test paths
  - component and variable cross-references
Restructuring

- Static analysis
  - Represent code as a semantic network
  - Not necessarily easy to read by humans
- Refined through successive simplifications
- Refined representation used to generate a structured equivalent body of code.

Chapter 10
Reverse Engineering

- Determine specification and design information from code.
- Recover engineering information.
- Store information in a repository.
Reengineering

- An extension of reverse engineering.
- Produce new software without changing the overall system function.