Training & Documentation

Reading: Chapter 10

Different Users

Types of training

- **User training**
  (what the system does)
- **Operator training**
  (how the system works)
- **Special training needs:**
  - new users vs. brush-up
  - infrequent vs. frequent users
Training Aids

- Documents
- Icons and on-line help
- Demonstrations and classes
  - Robotel
  - LAN School
  - Expert users

Documentation

- Consider the audience
- User manual
  - place in the documentation hierarchy
  - system purpose or objectives
  - Functions: description, inputs/outputs, etc.
- Operator manual
- General system guide (evaluation team)
- Tutorials and automated overviews
- Programmer’s guide

“Failure” Messages
Failure Messages

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.

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

Table 10.3. BASIC failure messages.

<table>
<thead>
<tr>
<th>Number</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Line buffer overflow. An attempt has been made to input a line that has too many characters.</td>
</tr>
<tr>
<td>24</td>
<td>Device time-out. The device you have specified is not available at this time.</td>
</tr>
<tr>
<td>25</td>
<td>Device fault. An incorrect device designation has been entered.</td>
</tr>
<tr>
<td>26</td>
<td>FOR without NEXT. A FOR statement was encountered without a matching NEXT statement.</td>
</tr>
<tr>
<td>27</td>
<td>Out of paper. The printer device is out of paper.</td>
</tr>
<tr>
<td>28</td>
<td>Unprintable error. A failure message is not available for the condition that exists.</td>
</tr>
<tr>
<td>29</td>
<td>WHILE without WEND. A WHILE statement was encountered without a matching WEND statement.</td>
</tr>
<tr>
<td>30</td>
<td>WEND without WHILE. A WEND statement was encountered without a matching WHILE statement.</td>
</tr>
<tr>
<td>31-40</td>
<td>Unprintable error. A failure message is not available for the condition that exists.</td>
</tr>
</tbody>
</table>
Maintenance

Reading: Chapter 11

Introduction

- Any change after delivery
- Software vs. Hardware Systems
  - SW: incorporate change
  - Dependency on the real world

Lehman’s program types

- $S$ - type Programs ("Specifiable")
  - problem can be stated formally and completely
  - acceptance: Is the program correct according to its specification?
  - This software does not evolve.
  - A change to the specification defines a new problem, hence a new program

- $P$ - type Programs ("Problem-solving")
  - imprecise specification of a real-world problem
  - acceptance: Is the program an acceptable solution to the problem?
    - Never mind whether the program is coded well (Lehman)

- $E$ - type Programs ("Evolutionary")
  - A system that becomes part of the world that it models
  - acceptance: depends entirely on opinion and judgment
  - This software is inherently evolutionary
    - embedded in the real world and changes as the world does
S-type

- The criterion of success in creating an S-type program is that it is correct in a strict mathematical sense
- Examples might be:
  - Space craft launch control, flight controls, air traffic control, MRI machine control, telephone switch, ABS control, safety critical systems, digital replacements for analog devices, micro programmed devices

S-systems

P-type

- Required to form an acceptable solution to a stated problem in the real world
  - For example, if a manufacturer stops supporting particular hardware or compilers, a program which depends on them may have to be ported to a new platform if it is not to become unusable.
  - Conversely, when new technical capabilities become available, they can trigger a reconsideration of design compromises, and possibly a redefinition of the problem abstraction.
P-Systems

E-Systems

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?
- 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?
- Is system performance still beyond prescribed constraints?
- Are system functions of limited usefulness?
Laws of software evolution

- Lehman: Laws of Software Evolution
  - Investigation started 1968
  - Based on measurement of a few (commercially-developed) systems, most notably IBM’s OS 360
  - Originally three laws, then five, now there are eight.
- Controversial as “laws”
  - Criticized for strong claims based on limited data.
  - Better term is “observations”
  - However, it’s pioneering work on software evolution and software engineering.

Continuing change: or less utility
Increasing complexity (entropy): 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

Four types of “maintenance”
- Corrective
  - Fix bugs & misunderstood requirements
- Adaptive
  - Respond to external changes (support other o.s./hardware)
- Perfective
  - Improve as-delivered software (new features, efficiency)
- Preventive
  - Before a failure occurs
### 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
  - \( k^* (m-k) + k^* (k-1)/2 \)
- Management priorities
- Morale

#### Technical problems
- Artifacts and paradigms
- Testing difficulties
  - No time
  - No data
  - No easy way to predict the effects of changes
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

Modeling Maintenance Effort

- COCOMO II:

\[ \text{Size} = \text{ASLOC} \times (\text{AA} + 0.4\text{DM} + 0.3\text{CM} + 0.3\text{iM})/100 \]

- AA = Assessment and Assimilation effort
- SU = Amount of Sw Understanding

Table 11.2. COCOMO II rating for software understanding.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Very low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very high</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very low cohesion, high coupling, spaghetti code</td>
<td>Moderately low cohesion, high coupling</td>
<td>Reasonably well structured; some weak areas</td>
<td>High cohesion, low coupling</td>
<td>Strong modularity, information-hiding in data and control structures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application clarity</th>
<th>No match between program and application</th>
<th>Some correlation between program and application</th>
<th>Moderate correlation between program and application</th>
<th>Good correlation between program and application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-descriptiveness</td>
<td>Obscure code and data; documentation missing or obsolete</td>
<td>Some useful commentary and headers; documentation missing or obsolete</td>
<td>Moderate level of code commentary and headers; documentation missing or obsolete</td>
<td>Good code commentary and headers; documentation up-to-date, well-organized, with design rationale</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SU increment</th>
<th>50</th>
<th>40</th>
<th>30</th>
<th>20</th>
<th>10</th>
</tr>
</thead>
</table>
Table 11.3. COCOMO II ratings for assessment and assimilation effort.

<table>
<thead>
<tr>
<th>Assessment and assimilation increment</th>
<th>Level of assessment and assimilation effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Basic component search and documentation</td>
</tr>
<tr>
<td>4</td>
<td>Some component test and evaluation documentation</td>
</tr>
<tr>
<td>6</td>
<td>Considerable component test and evaluation documentation</td>
</tr>
<tr>
<td>8</td>
<td>Extensive component test and evaluation documentation</td>
</tr>
</tbody>
</table>

Cyclomatic number

- Number of linearly independent paths through the code
- Control Flow Graph: Nodes and edges
  \[ e-n+2 \]
- Prepare test cases:
  - to force execution along each path

getSum: Control-Flow Graph

```java
1. int getSum(int i) {
   int j, sum;
2.   read j;
3.   sum = 0;
4.   while (i > 0 & i <= 10 || j<0) {
5.     if (j >0)
6.        sum = sum + j;
7.     i= i + 1;
8.   read j;
9. }
10.  return sum;
11.}
```

Some CFG edges are labeled (e.g., (4,5) and (5,7))
Some CFG edges are backedges (e.g., (7,8))