Software Maintenance (II)

Software Maintenance: Key challenges

(poor) quality of documentation

User demand for enhancements and extensions

Competing demands for maintainers’ time

Difficulty in meeting scheduled commitments

Turnover in user organizations

Software Maintenance: Key challenges

Limited understanding

47% of software maintenance effort devoted to understanding the software

System has “m” components, we need to change “k” of them. Thus there are k*(m-k) + k*(k-1)/2 interfaces to check for impact and correctness

50% of effort can be attributed to lack of user understanding (i.e., incomplete or mistaken reports of errors and enhancements)

Low morale
Software maintenance is regarded as less interesting than development

Main factors (1)

Application type
Systems with timing issues (real-time and highly synchronized);
Systems with rigidly defined data formats…

System novelty

Turnover and maintenance staff availability

System life span

Dependence on the changing environment
(P, E systems…)

Hardware characteristics

Main factors (2)

Design quality
Independent, cohesive components, well-defined architecture

Code quality

Documentation quality

Testing quality

Maintenance management

- Maintenance has a poor image amongst development staff as it is not seen as challenging and creative
- Maintenance costs increase as the software is maintained
- The amount of software which has to be maintained increases with time
- Inadequate configuration management often means that the different representations of a system are out of step

Maintenance cost factors

- Module independence
  - It should be possible to change one module without affecting others
- Programming language
  - High-level language programs are easier to maintain
- Programming style
  - Well-structured programs are easier to maintain
- Program validation and testing
  - Well-validated programs tend to require fewer changes due to corrective maintenance

Maintenance cost factors

- Documentation
  - Good documentation makes programs easier to understand
- Configuration management
  - Good CM means that links between programs and their documentation are maintained
- Application domain
  - Maintenance is easier in mature and well-understood application domains
- Staff stability
  - Maintenance costs are reduced if the same staff are involved with them for some time

Maintenance cost factors

- Program age
  - The older the program, the more expensive it is to maintain (usually)
- External environment
  - If a program is dependent on its external environment, it may have to be changed to reflect environmental changes
- Hardware stability
  - Programs designed for stable hardware will not require to change as the hardware changes

Maintenance metrics

Control complexity

Can be measured by examining the conditional statements in the program

Data complexity

Complexity of data structures and component interfaces.

Length of identifier names

Longer names imply readability

Program comments

more comments mean easier maintenance

Maintenance metrics

Coupling

How much use is made of other components or data structures

Degree of user interaction

The more user I/O, the more likely the component is to require change

Speed and space requirements

Require tricky programming, harder to maintain

Process metrics

- Number of requests for corrective maintenance
- Average time required for impact analysis
- Average time taken to implement a change request
- Number of outstanding change requests
- If any or all of these is increasing, this may indicate a decline in maintainability

Program Structure Improvement

- Maintenance tends to corrupt the structure of a program. It becomes harder and harder to understand

- The program may be automatically restructured to remove unconditional branches

- Conditions may be simplified to make them more readable

Spaghetti Logic

```
Start:    Get (Time-on, Time-off, Time, Setting, Temp, Switch)
          if Switch = off goto off
          if Switch = on goto on
          goto Cntrld
off:  if Heating-status = on goto Sw-off
          goto loop
on:  if Heating-status = off goto Sw-on
          goto loop
Cntrld:   if Time = Time-on goto on
          if Time = Time-off goto off
          if Time < Time-on goto Start
          if Time > Time-off goto Start
          if Temp > Setting then goto off
          if Temp < Setting then goto on
Sw-off:   Heating-status := off
          goto Switch
Sw-on:    Heating-status := on
Switch:   Switch-heating
loop:     goto Start
```

Structured Control Logic

```
loop
    -- The Get statement finds values for the given variables from the system's
-- environment.
    Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
    case Switch of
        when On => if Heating-status = off then
                        Switch-heating ; Heating-status := on ;
                    end if ;
        when Off => if Heating-status = on then
                        Switch-heating ; Heating-status := off ;
                    end if;
        when Controlled =>
            if Time >= Time-on and Time < = Time-off then
                if Temp > Setting and Heating-status = on then
                    Switch-heating; Heating-status = off;
                elsif Temp < Setting and Heating-status = off then
                    Switch-heating; Heating-status := on ;
                end if;
            end if ;
    end case ;
end loop ;
```

Maintainability and readability (textual products)

Gunning Fog index

$$\boxed{\mathrm{F} = 0.4[\frac{\text{number of words}}{\text{number of sentences}} + \text{percentage of "complex" words of 3 or more syllabes}]}$$

Select a passage of ~ 1,000 words

Examples:

Readers Digest 8-9
Time Magazine ~ 11
technical documentation 10-15
>13 hard to read

Measuring maintenance characteristics

Note: maintainability is not restricted to code – could apply to different software products (specification, design, tests, documentation)

External view of maintainability
expressed as <u>mean time to repair</u>
(we should know the time at which the problem is reported, time required to
analyze the problem, to specify which change are to be made, time needed to
Make the change, test the change, document the change)

Internal view of maintainability
Complexity of code,
code size,
fan-in and fan-out characteristics → [fan-in * fan_out]^2
quality of documentation…

Traceability

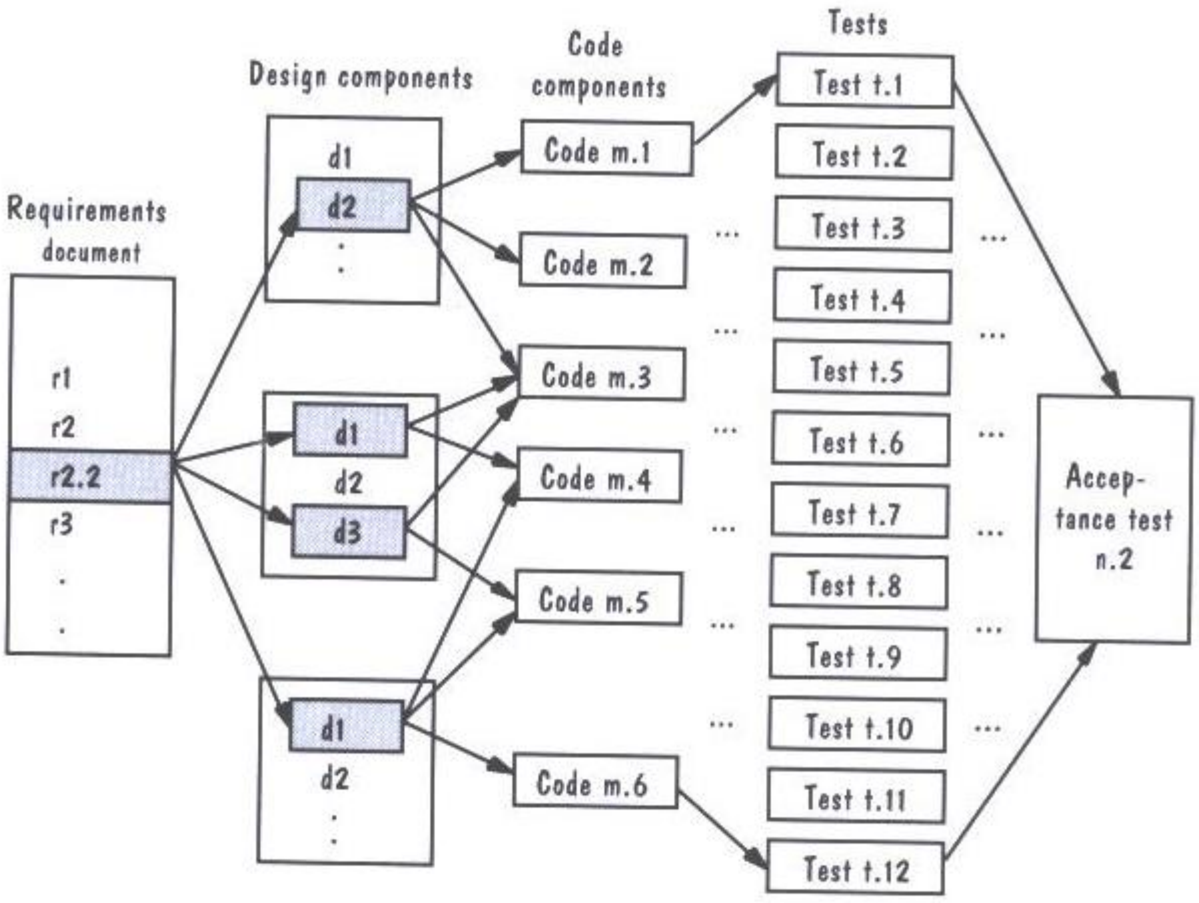
workproduct: any development artifact whose undergoes change
(say, requirements, design, code components, test cases, documentation)

IMPACT OF CHANGE for all workproducts

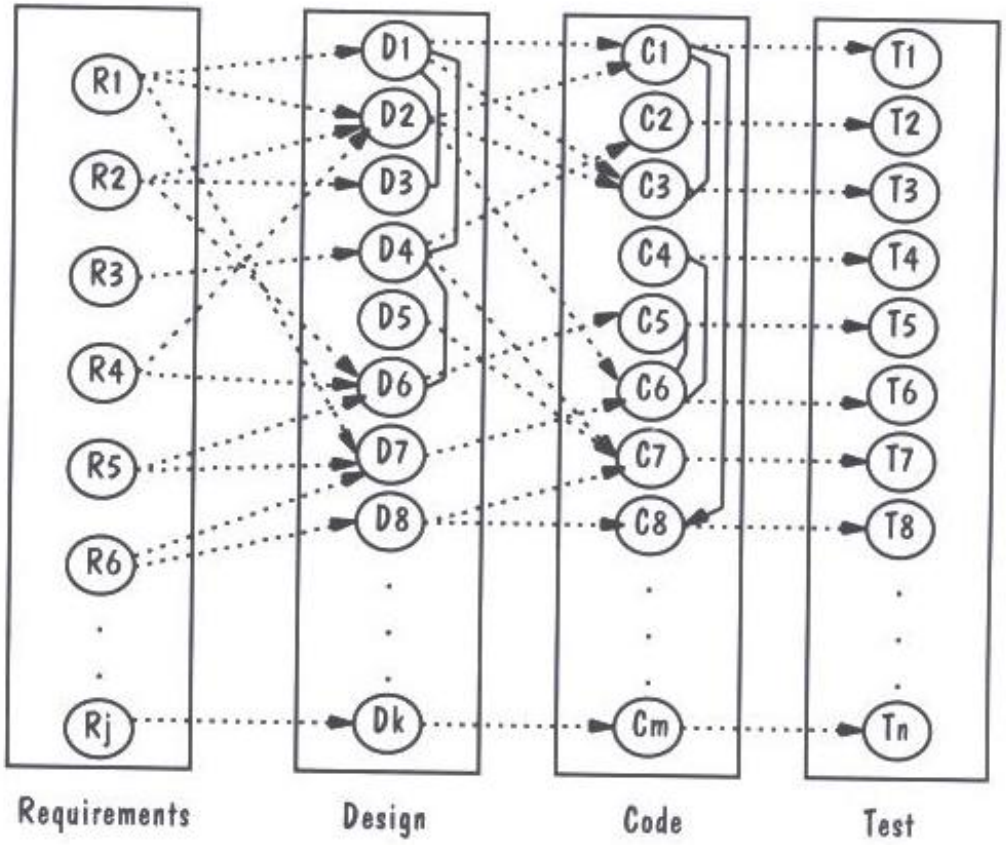
Vertical traceability : expresses the relationships among the parts of the workproduct (e.g., interdependencies among system requirements); product view of change

Horizontal traceability: expresses the relationships of the components across the collections of workproducts; process view of change

Horizontal traceability: example



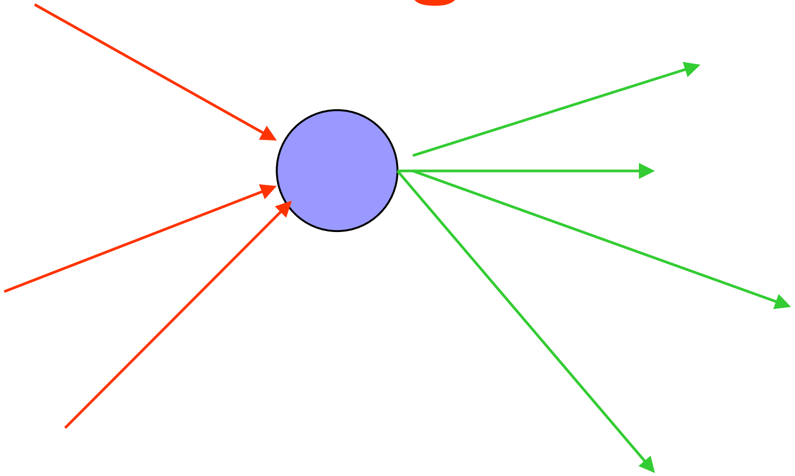
Traceability graph



Vertical traceability ——————

Horizontal traceability:........................

Traceability graph- evaluation of risk of change



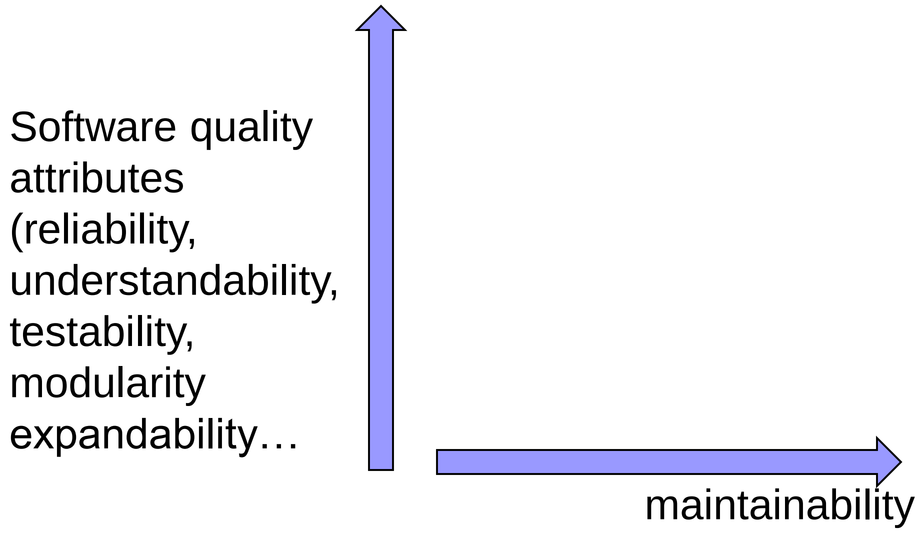
in-degree [the number of edges the node is a destination of]
out-degree [the number of edges the node is a source of]
Complexity measure **of node evaluated before and after the change** (e.g., cyclomatic complexity)

Risk = f(in-degree, out-degree, complexity measure of node)

Main causes of unmaintainable software

National Institute of Science & Technology (NIST)

- **Poor software design**
- **Poorly coded software**
- **Software designed for outdated hardware**
- **Lack of common data definitions**
- **Use of multiple languages in one program**
- **Grown software inventory**
- **Excessive resource requirements**
- **Inadequate documentation**
- **Inadequate user interface**
- **Lack of highly skilled staff**



Software quality attributes (reliability, understandability, testability, modularity expandability…

maintainability