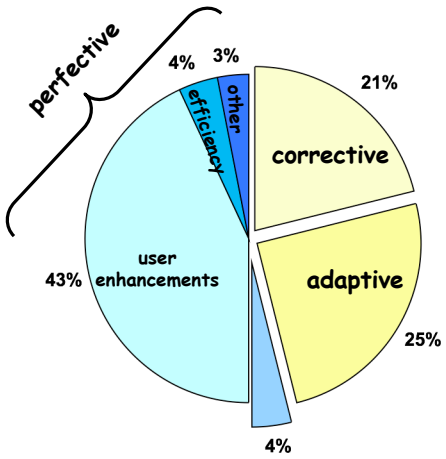
Lecture 6: Software Re-Engineering

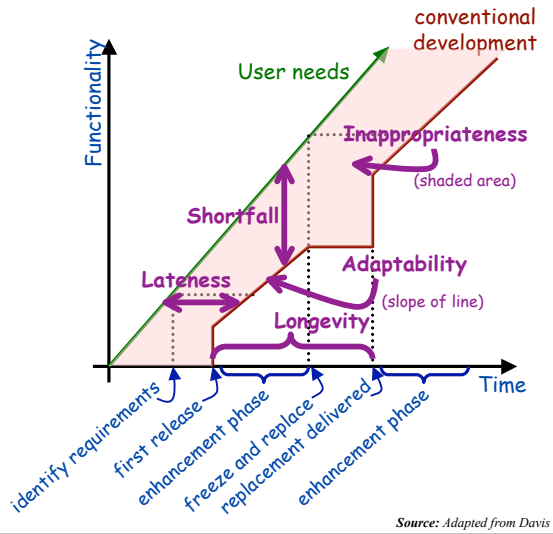
- → Cost of Software Maintenance
- → Challenges of Design Recovery
- → What reverse engineering tools can and can't do
- → Hints on abstraction and design recovery

Software Evolves Continuously



Data from:
van Vliet, H., *Software Engineering: Principles and Practices*, Wiley 1999, p449

User requirements always grow



Source: *Adapted from Davis 1988, pp1453-1455*

Software Geriatrics

Causes of Software Aging

- Failure to update the software to meet changing needs
  - Customers switch to a new product if benefits outweigh switching costs
- Changes to software tend to reduce coherence & increase complexity

Costs of Software Aging

- Owners of aging software find it hard to keep up with the marketplace
- Deterioration in space/time performance due to deteriorating structure
- Aging software gets more buggy
  - Each "bug fix" introduces more errors than it fixes

Ways of Increasing Longevity

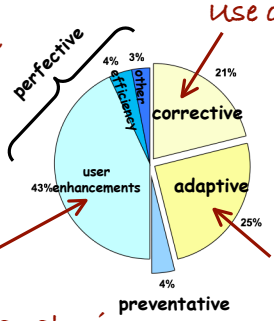
- Design for change
- Document the software carefully
- Requirements and designs should be reviewed by those responsible for its maintenance
- Software Rejuvenation…

Source: *Adapted from Parnas, "Software Aging" 1996*

Reducing Maintenance Costs

General
Modular structure
Comprehensibility
Good documentation

Higher quality code
Better testing (verification)
Use of standards



Platform independence
Design for change
Good architecture

Better requirements analysis
prototyping, iterative development
Design for change

E.g. The Altimeter Example

```
IF not-read1(V1) GOTO DEF1;
display (V1);
GOTO C;
DEF1: IF not-read2(V2) GOTO DEF2;
display(V2);
GOTO C;
DEF2: display(3000);
C:
```

```
if (read-meter1(V1))
  display(V1);
else {
  if (read-meter2(V2))
    display(V2);
  else
    display(3000);
}
```

Questions:

- Should you refactor this code?
- Should you fix the default value?

Source: Adapted from van Vliet 1999

Why maintenance is hard

Poor code quality

- opaque code
- poorly structured code
- dead code

Lack of knowledge of the application domain

- understanding the implications of change

Lack of documentation

- code is often the only resource
- missing rationale for design decisions

Lack of glamour

Source: *Adapted from van Vliet 1999*

Rejuvenation

Reverse Engineering

- Re-documentation (same level of abstraction)
- Design Recovery (higher levels of abstraction)

Restructuring

- Refactoring (no changes to functionality)
- Revamping (only the user interface is changed)

Re-Engineering

- Real changes made to the code
- Usually done as round trip:
  - design recovery -> design improvement -> re-implementation

Source: *Adapted from van Vliet 1999*

Program Comprehension

During maintenance:

- programmers study the code about 1.5 times as long as the documentation
- programmers spend as much time reading code as editing it

Experts have many knowledge chunks:

- programming plans
- beacons
- design patterns

Experts follow dependency links

- …while novices read sequentially

Much knowledge comes from outside the code

Source: *Adapted from van Vliet 1999*

Example 1

What does this do?

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    if (A[i,j]) {
      for (k=0; k<n; k++) {
        if (A[j,k])
          A[i,k]=true;
      }
    }
  }
}
```

Source: *Adapted from van Vliet 1999*

Example 2

```
procedure A(var x: w);
begin
  b(y, n1);
  b(x, n2);
  m(w[x]);
  y := x;
  r(p[x]);
end;
```

```
procedure change_window(var nw: window);
begin
  border(current_window, no_highlight);
  border(nw, highlight);
  move_cursor(w[nw]);
  current_window := nw;
  resume(process[nw]);
end;
```

Source: Adapted from van Vliet 1999

What tools can do

Reformatters / documentation generators

- Make the code more readable
- Add comments automatically

Improve Code Browsing

- E.g visualize and traverse a dependency graph

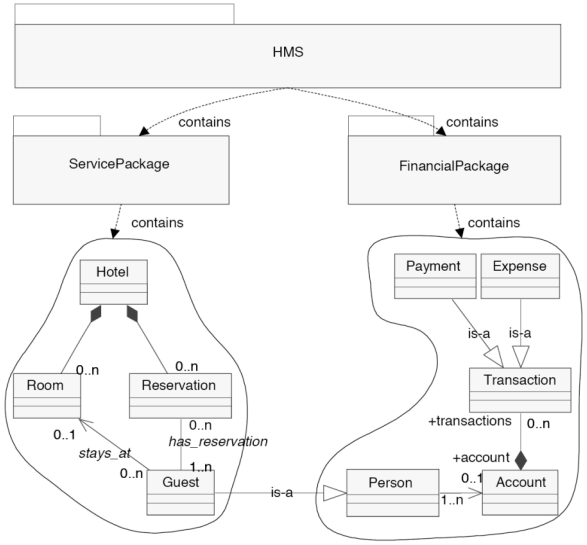
(simple) Code transformation

- E.g. Refactoring class browsers
- E.g. Clone detectors

(simple) Design Recovery

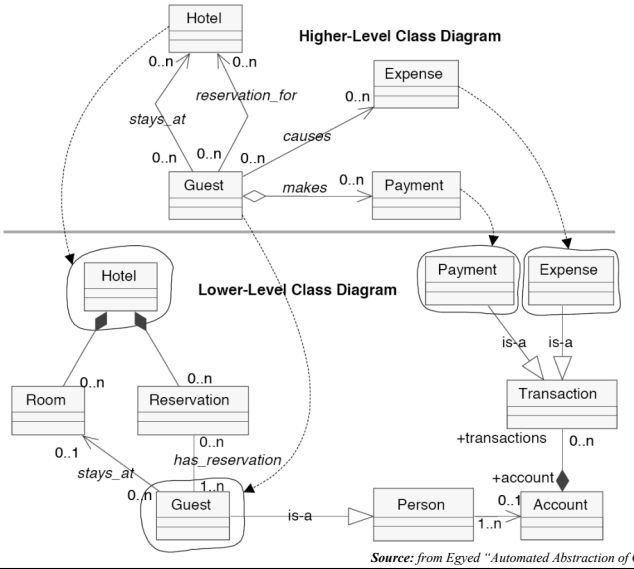
- E.g. build a basic class diagram
- E.g. use program traces to build sequence diagrams

Package Decomposition



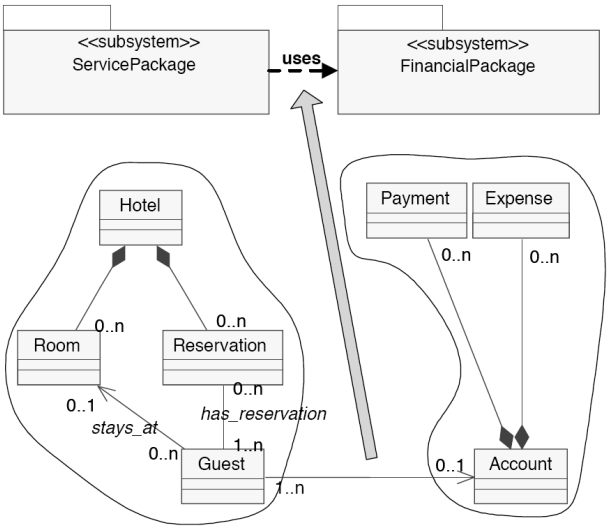
Source: *from Egyed "Automated Abstraction of Class Diagrams, TSE 2002*

Class Abstraction



Source: *from Egyed "Automated Abstraction of Class Diagrams, TSE 2002*

Finding Dependencies



Source: *from Egyed "Automated Abstraction of Class Diagrams, TSE 2002*