Major Problem in Software

Dependency

Dependency Management

- What is dependency management (DM)?
- What bearing does DM have on software?
- What is the result of poor DM?
- What is the advantage of good DM?

What is dependency management?

- As interdependencies increase, features like reusability, flexibility, and maintainability decrease.
- Dependency management is controlling interdependencies.

What bearing does DM have on software?

- Coupling and cohesion are the eternal concerns of software development
- One can say that OO is just a set of tools and techniques for Dependency Management

What is the penalty for practicing poor DM?

A system with poor dependency structure will typically exhibit these four negative traits:

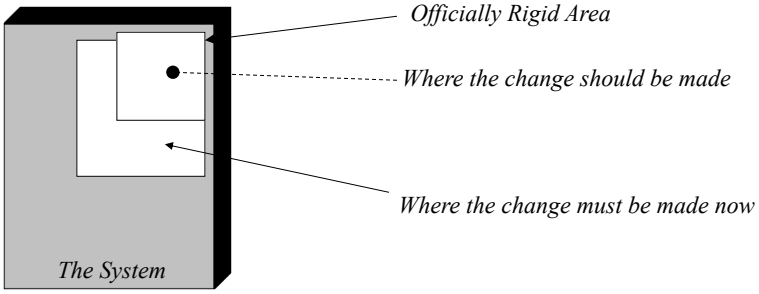
- It is rigid
- It is fragile
- It is not reusable
- It has high viscosity

It is Rigid

Rigidity is the inability to be changed

- The impact of a change cannot be predicted
- If not predicted, it cannot be estimated
- Time and cost cannot be quantified
- Managers become reluctant to authorize change

Changes with Rigidity



Are we containing risk, or spreading rot?

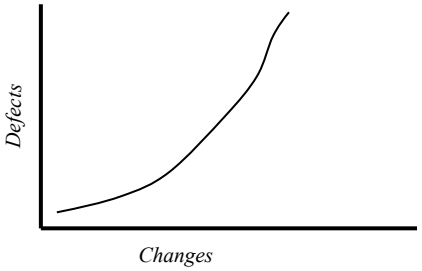
It is Fragile

Software changes seem to exhibit non-local effects

- A single change requires a cascade of subsequent changes
- New errors appear in areas that seem unconnected to the changed areas
- Quality is unpredictable
- The development team loses credibility

Increasing Risk

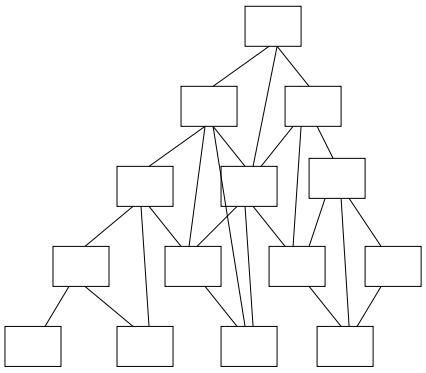
Defects vs. Cumulative Modifications



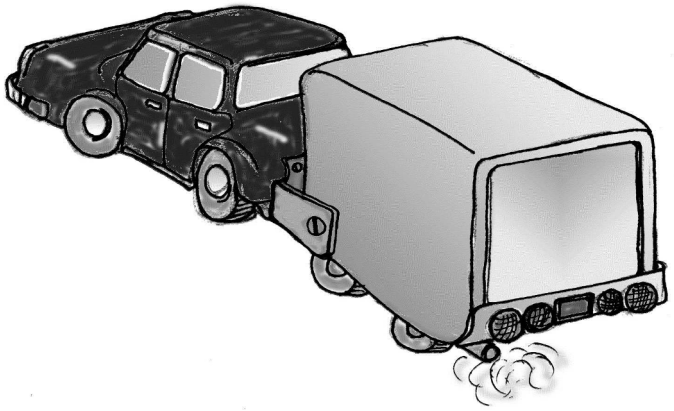
Systems tend to become increasingly fragile over time. Intentional, planned partial rewrites may be necessary to sustain growth and maintenance.

It is not reusable

- Desirable parts of the design are dependent upon undesirable parts
- The work and risk of extracting the desirable part may exceed the cost of redeveloping from scratch



The Trailer



It has high viscosity

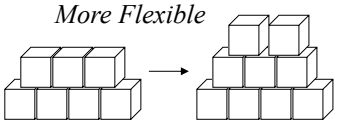
Viscosity is resistance to fluid motion

- When the "right changes" are *much more difficult* than hacking, the viscosity of the system is high.
- Over time, it will become increasingly difficult to continue developing the product.

Benefit of good DM

Interdependencies are managed, with firewalls separating aspects that need to vary independently.

More Flexible



Less fragile, the bugs are boxed in

Easier to reuse

Fewer Trailers

Easier to make the right change

Slow the rot

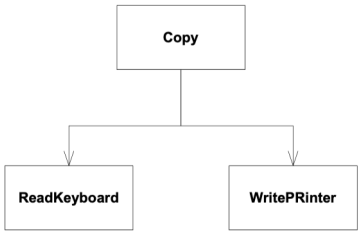
What causes “Code Rot”?

It’s been blamed on stupidity, lack of discipline, and phases of the moon, but...

A case study “The Copy Routine”

First Version

All designs start well




```
void copy(void)
{
     int ch;
     while( (ch=ReadKeyboard()) != EOF)
          WritePrinter(ch);
}
```

The program is an overnight success!
How could it be more simple, elegant, and maintainable?

Second Version

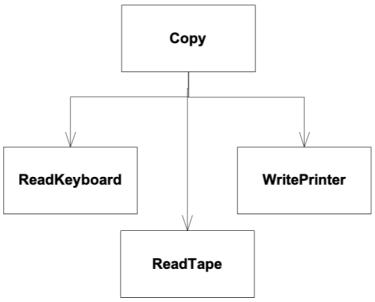
Oh, no! Nobody said the requirements might change!

- We sometimes want to read from paper tape reader.
- We could put a parameter in the call, but we have hundreds of users already!
- No big deal, this is just an exception… we can make it work.

Second Version Design

```
bool GtapeReader = false; // remember to clear

void copy(void)
{
    int ch;
    while( (ch=GtapeReader ? ReadTape() : ReadKeyboard()) != EOF)
        WritePrinter(ch);
}
```



Third Version

How unexpected! Requirements changed again!

It seems that sometimes we need to write to a paper tape punch. We've had this problem before, and just added a flag. Looks like it should work again.

```
bool GtapeReader = false;
Bool GtapePunch = false;
// remember to clear

void copy(void)
{
    int ch;
    while( (ch=GtapeReader ? ReadTape() : ReadKeyboard()) != EOF)
        GtapePunch ? WritePunch(ch) : WritePrinter(ch);
}
```

Example of a Good Design

First and only version.

```
void Copy()
{
   int c;
   while( (c=getchar()) != EOF)
      putchar(c);
}
```

But wait! Aren't we supposed to using OO design? This isn't OO is it?

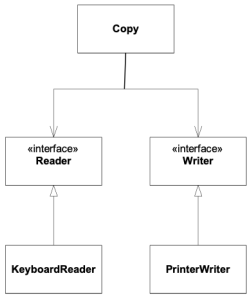
...is it?

It is a small program based on abstractions!

- FILE is an abstraction
  - It represents some kind of byte stream
  - It has many variations
- It has methods
  - Read, Write, getchar, putchar, etc
  - The methods are *dynamically* bound

FILE is a class, just implemented differently.

Rephrased in OO




```
interface Reader
{ char read(); }

interface Writer
{ void write(char c); }

public class Copy
{
   private Reader itsReader;
   private Writer itsWriter;

   Copy(Reader r, Writer w)
   {
       itsReader = r;
       itsWriter = w;
   }
   public void copy()
   {
       int c;
       while( (c==itsReader.read()) != EOF )
           itsWriter.write(c);
   }
}
```


Dependency Management Review

- Why do programs tend to rot over time?
- What is dependency management?
- What are four qualities of good designs?
- Are OO programs always simpler than non-OO versions?
- Why would anyone want to use a paradigm that may result in more sophisticated designs?