ζ How Software Components Grew Up and Conquered the World

ζ Butler Lampson

ζ For many years programmers have dreamed of building systems from a library of reusable software components together with a little new code. The closest we've come is Unix commands connected by pipes. I'll discuss the fundamental reasons why software components of this kind haven't worked in the past and are unlikely to work in the future. Then I'll explain how the dream has come true in spite of this failure, and why most people haven't noticed.

How Software Components Grew Up and Conquered the World

Butler Lampson

Microsoft

May 21, 1999

The software crisis

- It's always with us. Why?
  - Moore's law means always trying new things
  - Complexity moves into software
  - Can't find the limits except by trial and error
    - "A man's reach should exceed his grasp, or what's a heaven for."
      – *Browning*
- Can components make a difference?
  - PITAC says yes

Progress?

- Programming from scratch
- Modifying existing code

A little

- Building on big components
  - "White-collar" computing
- High-level programming
  - Declarative
  - Domain-specific
  - Examples: spreadsheets, SQL

A lot

The component dream

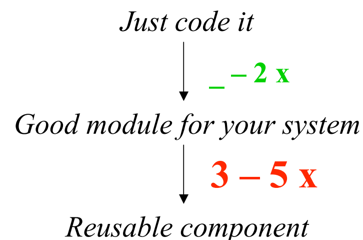
- A library of tested, documented components
- To build your system, you take down a couple of dozen components and glue them together.

Why components don't work

- No business model
- Cost to understand and use
- Conflicting world views

No business model

- A reusable component costs at least 3x as much as a good module for your system.
  - Generality
  - Simplicity
  - Customization
  - Testing
  - Documentation
  - Stability



- Who will pay?
  - The market won't support this kind of development

Cost to understand

- To use a component, must understand its behavior
  - Resource consumption
  - Exceptions
  - Customization
  - Bugs
  - Workarounds when it doesn't do what you want
- One measure: ratio of spec size to code size
- The alternative: re-implement it yourself

Conflicting world views

- A component interface embodies a view of the world:
    - Data types
    - Resource consumption
    - Memory allocation
    - Exception handling
    - …
- When you put 10 world views together, the result is a mess.
  - No one is responsible for design integrity, or for the whole thing working
- Interactions are N^2 ($N \log N$ if you're lucky)

Components that do work

- Unix pipes
  - Built by Bell Labs Unix group
  - Very simple interface: char stream
  - Unencoded: just words and lines
- Libraries (math and graphics)
  - Take a long time to perfect
  - Clean mathematical model
  - World view inherited from math, and library usually done by a single group
- Lesson: It works in simple, narrow domains

Components in PC's

- PC's are built from (hardware) components
  - Processor chip
  - DRAM SIMM
  - Disk + driver
  - Display monitor + driver
  - . . . + driver
- Why does it work?
  - Only a few components work well
    - These have very clear interfaces
    - Odd-ball ones have lots of problems
  - Microsoft is responsible for integrity

Community makes it easier

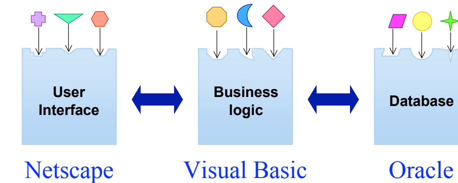
- Includes the component builders and its users
  - Research or engineering community
- People work for status in the community
- Shared context that everyone understands
- Common world view

Good things, but not reusable components

- COM and Corba
  - These are ways to run components
- Visual Basic components and ActiveX.
  - You can use a couple, but not more
- Modules with clean interfaces
  - These are necessary, but not sufficient

Big components work

- They are so huge that you only use 3 of them.
  - 5-20 M lines
  - Operating system: Linux, Windows
  - RAD environment: Visual Basic, Java
  - Browser: Netscape, Internet Explorer
  - Database: Oracle, DB2, SQL Server



Why big components work

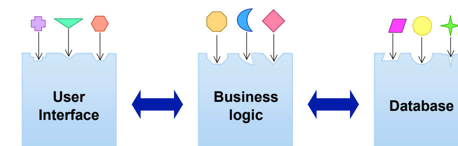
Business model
Cost to understand
World view

- You can sell them
  - Customer only has to buy a few
- The spec is much smaller than the code
  - and you can customize them
- They impose their own world view
  - Vendor provides design integrity inside
  - $N^2 = 9$

- Moore's law –cycles and storage to burn

Customization is critical

- OS's have applications, scripts
- Browsers have scripts, Java, plug-ins, DHTML
- RDBs have SQL



Burning cycles

- Is it bad engineering? No.
- Engineer's job is greatest customer value
  - Time to market
  - Flexibility
  - Total cost of ownership
- Today's PC = 1000 x Alto. Where did it go?
  - Deliver lots of features quickly
    - Can't have first-class design everywhere
    - Trade resources for time to market
  - Integration
  - Support lots of hardware
  - Backward compatibility

What else could work?

- Code manipulation
- Specs with teeth
  - Encapsulation
- Declarative programming

Code manipulation

- Copy a module and change it
  - The most flexible way to customize
- Tools to support this
  - Specs help in understanding the code
  - Checks (types, assertions, …) catch mistakes
  - Analyzers reveal structure
- The true destiny of open source?

Specs with teeth

- Documentation with checks that:
  - The component matches the documentation
  - The use is legal
- Checking ensures correct documentation
  - More likely to pass the coffee stain test
- Prototype: type checking

Specs with teeth

- Create/generate vs enforce
  - Create: Stubs, transactions, GC
  - Enforce: Sandbox, Purify, types
- Static vs dynamic check
  - Static: Types, PCC
    - No dynamic error can occur
    - There's no debugger in Peoria. *Morris*
    - Well-typed programs don't go wrong. *Milner*
  - Dynamic: Sandbox, XML

Encapsulation

- Sandbox
  - Operating system, Java, SFI, information flow
- Code modification
  - SFI, Eraser, Purify
- Transactions
  - Automatic concurrency, fault tolerance, resource management
- Replicated state machines
  - Automatic fault tolerance

Declarative programming

- Examples
  - Spreadsheets
  - SQL
  - Mathematica
  - YACC
- Advantages
  - Closer to intent, hence faster and more reliable
  - More analysis and optimization—parallel SQL
- High-level programming?

Summary

- The component dream can't work
- Big components do work
  - Use only 3
  - Customize
- What else?
  - Encapsulate
  - Declare