COMPLEXITY

![Graph showing increasing complexity over time. The y-axis represents complexity per developer, and the x-axis represents time.]
NSB REDUX

• Confusions over accidental complexity
• Emergence of OOP
• “The best way to address the complexity of software is to not build it at all”
  • Several NSB responses highlight software reuse
    • As a means to reduce complexity
    • As a means to improve productivity
    • As a means to increase reliability
SOFTWARE REUSE

• Domain-specific component markets
  • Populated by carefully created reusable components
• New features are added by dropping in components
  • Accounts for 53% of reuse at NASA [Selby 2005]
• Three main impediments:
  • High up-front cost [Gaffney 1992, ICSE]
  • Library scaling problem [Biggerstaff 1994, ICSR]
  • Architectural mismatch [Garlan et. al. 1995, IEEE Software]
ECONOMICS

• Budgets are drawn up annually
  • Heavy emphasis on the current quarter
• Reusable software is:
  • ~Twice as expensive [Gaffney 1992, ICSE]
  • ~Three times as expensive [Brooks 1975]
• Requires careful forethought to determine what software will be reused and whether any savings outweigh extra costs
• What is the benefit to the customer?
LIBRARY SCALING

- Two extremes:
  - Large, feature-laden, components
  - Small, simple, components
- Adapting large components to a system can be difficult
- The effort of adapting a small component might outweigh any benefits of reuse in the first place
ARCHITECTURAL MISMATCH

• Even reusable code makes some assumptions about *how* it should be reused; these assumptions are often implicit
  • Explicit assumptions are often easy to identify:
    • Programming language
    • Libraries & frameworks
  • Implicit assumptions are harder to spot:
    • Topology assumptions
    • Protocols of use
• Implicit assumptions are often not documented because the original developer may not have considered them constraints
AN ALTERNATIVE REUSE APPROACH

Bug: UltiGPX should visualize elevation changes in tracks using a profile view.

I wish UltiGPX showed me how my elevation has changed...

UltiGPX
MOTIVATION

Azureus

UltiGPX

1) Plan

2) Enact
MANUAL REUSE APPROACH

- Easy to get discouraged
- Difficult to modify earlier decisions
- Easy to attempt infeasible tasks
- Piecemeal nature diverts attention from core technical difficulties
PRAGMATIC REUSE

• White-box reuse
  • Code Scavenging [Krueger 1992, ACM Computing Surveys]
    • Ad hoc nature increases risk of bad decisions
    • Adaptation expensive and overwhelming
  • Industrially effective
    • Effective reuse approach [Frakes 1995, CACM]
    • Common risk-aversion practice [Cordy 2003, IWPSE]
    • Replicate & specialize [Kapser & Godfrey, 2006, WCRE]
PRAGMATIC REUSE PROCESS

Identify starting point

Plan task / triage dependencies

Enact plan

Evaluate reused code

Abort task

Re-implement feature
PLANNING A TASK

• 4 main kinds of decisions:
  • Common
  • Accept
  • Reject
  • Remap

Legend
O Method or Field
→ Call or Reference
TRANSITIVE IMPLICATIONS
ENACTMENT PROCESS

Existing Code

1) Extraction

2) Integration

Accepted Code
Rejected Code
Remapped Code

Developer’s System

1) Extraction
2) Integration
Developers using Gilligan significantly faster
Repeated Measures ANOVA (F(1,14)=5.1, p=0.04)
**EFFORT-BASED CASE STUDY**

Automatically resolving low-level compilation errors enables the developer to focus on higher-level mismatch.

<table>
<thead>
<tr>
<th>Case</th>
<th>Manual</th>
<th>Gilligan</th>
<th>Decision Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>60</td>
<td>2</td>
<td>97%</td>
</tr>
<tr>
<td>T2</td>
<td>25</td>
<td>4</td>
<td>84%</td>
</tr>
</tbody>
</table>
PR SHORTCOMINGS

• Pragmatic reuse tasks are fraught with problems
  • Reused code is less-understood
  • Tracking and merging changes difficult
  • Often associated with bad practice
• Rely on software engineers to make the right decisions about downsides and benefits of these tasks

• Questions?