Reuse and Components (2)

CS3 / SEOC1

Note 13

Reuse: The Success Stories (1): EDS (Electronic Data Systems) early 1990's

- Smalltalk programmers given same specifications and test suites as earlier team who produced PL/1 system
- PL/1:
  - 265,000 SLOC (Source lines of code)
  - 152 staff months
  - 19 months to develop
- Smalltalk:
  - 22 SLOC
  - 10 staff months
  - 3.5 months to develop

Reuse: The Success Stories (2): NASA SEL (Software Engineering Laboratory)

- routinely 75% or higher reuse
- development time reduced by 90%
  - 58,000 hours for application development,
  - recently, with reuse, reduced to approx. 6,000

Whatever Happened to Reuse?

- EDS experience not repeated
  - common arguments: wrong language, "not invented here" syndrome, lack of management support
  - numbers inflated?
- NASA success hard to transfer: *domain* and *economics*
- specialised problem domain (spacecraft flight dynamics)
- very high initial investment:
  - 36,000 hours for domain analysis
  - 40,000 hours to develop components
  - design of reusable asset library starts 1992
  - first application using library in 1995
  - SEL estimates library development costs recouped by 4th mission

Reuse Pitfalls (1)

- Underestimating the difficulty of reuse
  - software must be more general
  - similarities among projects often small
  - much of what is reused is already provided by OS
  - universe in constant flux (hardware, software, environment, requirements, expectations, ...)
- Having or setting unrealistic expectations
  - OO reuse overly "hyped"
  - "Software is not built from Lego$_{TM}$ blocks" – *Alexander Ran*
  - reuse domain may be unrealistic
  - expectations for reuse outstrip skills of developers

Reuse Pitfalls (2)

- Being too focused on code reuse
  - focus on code reuse as end, not means
  - "'lines of code reused" metric meaningless
  - design reuse often neglected in favour of code reuse
  - too little abstraction at framework level
- Not investing in reuse
  - reuse costs: time and money in development, analysis, design, implementation, testing, ...

Reuse Pitfalls (3)

- Generalising after the fact
  - designs often migrate from general to specific during development
  - system not designed with reuse in mind (cf. code reuse versus code salvage)
- Allowing too many connections
  - coupling unaviodable, but must be very low to permit reuse
  - Circular dependencies also problemmatic – where to break the chain?

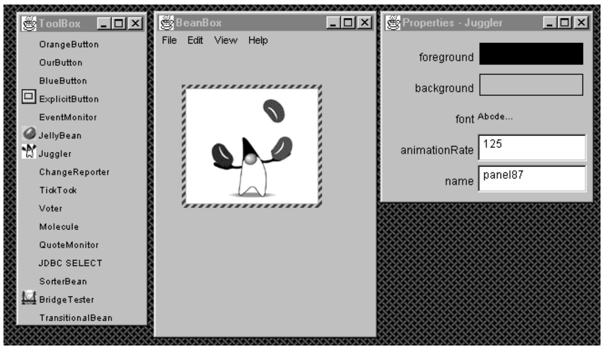
Difficulties with Component Development

- Economic: small business do not have long term stability and freedom required
  When you're up to your ass in alligators...
- Where is the 3rd party component market?
  - effort in (re)using components
  - cross-platform and cross-vendor compatibility
  - many common concepts, few common components
  - some successes: user interfaces, data management, thread management, data sharing between applications
  - most successful: GUI's and data handling (e.g. Abstract Data Types)

Components in Java

- JavaBeans
  - visual composition of components
  - builder introspection of Bean features (properties, methods, events)
  - composition of Beans into applets, applications or other Beans
- ADT's: `java.util.*` library
- GUI's: The Java Foundation Classes (JFC)

The Java "BeanBox" component builder



Java Foundation Classes

- History: AWT ("Abstract Window Toolkit" – 1995), JFC (1997 – Swing)
- all components are JavaBeans
- lightweight UI framework
  - peerless emulation versus layered ("peer") toolkit model
  - cross platform (no native code)
- pluggable look and feel
- no framework lock-in ("easily" compatible with 3rd party components)
- subclasses are fully customisable and extendible (according to Sun)

A JFC Example (taken from Sun's on-line Swing Tutorial)

```
import java.swing.*;
public class SwingApplication {
  public static void main(String[] args) {
    JFrame frame =
          new JFrame("SwingApplication");
    frame.getContentPane().add(contents,
          BorderLayout.CENTER);

    frame.addWindowListener(...);
    frame.pack();
    frame.setVisible(true);
  }
}
JButton button =
          new JButton("A button!");
button.setMnemonic(KeyEvent.VK_I);
button.addActionListener(...);
}
```

JFC Example (2)

```
final JLabel label =
          new JLabel(labelPrefix + "0  ");
label.setLabelFor(button);
label.setText(labelPrefix + numClicks);

JPanel pane = new JPanel();
pane.setBorder(BorderFactory.
          createEmptyBorder(...));
pane.setLayout(new GridLayout(0, 1));
pane.add(button);
pane.add(label);

button.addActionListener(
  new ActionListener(){
   public void actionPerformed(
          ActionEvent e){
    numClicks++;
    label.setText(labelPrefix + numClicks);
  } });
}
```


Summary

- many reuse kinds – software and knowledge
- component reuse is a form of software reuse
  - encapsulation, high cohesion, specified interfaces, explicit context dependencies
  - component development requires significant time and effort
  - as does component use
- employing reuse requires management
- Java (potentially) supports cross-platform component reuse
- component reuse has been successful for interfaces and data handling
- JFC and `java.util.*` classes exemplify this