Killer “Killer Examples” for Design Patterns

Carl Alphonce
alphonce@cse.buffalo.edu
Department of Computer Science & Engineering
University at Buffalo, SUNY

Adrienne Decker
adrienne@cse.buffalo.edu

Michael Caspersen
mec@daimi.au.dk
Department of Computer Science
University of Aarhus
Common themes of this session

• Patterns, patterns everywhere
  – in what we teach (first two papers of session deal with design patterns)
  – in how we teach (last paper of session deals with pedagogical patterns)

• Patterns work best when they support each other
What is a Design Pattern?

- Who does *not* want me to explain what a design pattern is?
- According to GoF:
  
  *Design Patterns describe simple and elegant solutions to specific problems in object-oriented software design.*

  *[Preface]*

- Another description
  
  *Design Patterns are “best-practices” solutions to common software design problems.*
“Killer Examples” for Design Patterns workshops

- Gathers “Killer Examples” from industry and academia
- Held at OOPSLA:
  - 2006 in Portland
  - 2005 in San Diego
  - 2004 in Vancouver
  - 2003 in Anaheim
  - 2002 in Seattle
- In this presentation we share some lessons that have come out of the workshop series
Why teach design patterns?

• Students need to learn concepts/skills with staying power.
• In other words, don’t focus on tools (i.e. languages, technologies), but what we can do with the tools.
• Software correctness is important, but so are other qualities, such as scalability, extensibility, flexibility, and robustness.
Challenges: student preconceptions

• Students tend to focus only on input-output behavior of their programs.
• Students tend not to focus on the quality of the solutions they come up with.
  – grading can reinforce this idea
  – nature of assignments can also reinforce this
• Students tend to have a very skewed view of the software development process (e.g. linear “poof” process).
Challenges: student preconceptions

• Beginning students often do not believe design patterns are used in “real world” software design.
  – they are surprised to learn object-orientation and design patterns can actually be (and are) used in safety-critical embedded military applications, for example
Challenges: dispelling the misconceptions

• Examples which benefit from application of design patterns tend to be rich in structure and complexity.

• These examples therefore naturally tend to be less accessible to students than simpler and smaller “textbook” examples.
Challenges:
Where do good examples come from?

- Examples which faculty construct lack “street-cred”

- Students want to see “real-world” application of the ideas they are learning. Otherwise they are too easily dismissed as “ivory tower” examples.
Lessons learned: Context

• Patterns cannot effectively be taught in isolation: context of problem gives motivation.

• Patterns must be presented in a context which clearly demonstrates the usefulness of the pattern in comparison to the same software built without them.
Lessons learned: Accessibility

• Students must readily grasp the context of the problem (e.g. an interactive program guide for cable or satellite TV).

• Spending too much time understanding the domain of a problem distracts from course goals.
Lessons learned: Real-world

• Patterns are mined from real-world code.
• Examples must reflect this fact.
• This is an important connection and motivation for studying design patterns for many students.
Lessons learned: Clear benefits

- Benefits which accrue due to use of patterns must be clearly spelled out to students.

- They must see how design pattern use improves the readability, scalability, flexibility, etc.
Pedagogy

• Intra-pattern considerations

• Inter-pattern considerations
Pedagogy

• Intra-pattern considerations
  – use it
  – conceptualize it
  – build it
  – analyze/study high quality code

• Inter-pattern considerations
Pedagogy

• Intra-pattern considerations
  – use it
  – conceptualize it
  – build it
  – analyze/study high quality code

• Inter-pattern considerations
  – design and construct software solutions
  – evaluate
Killer “Killer Examples”
Three representatives from the workshops

• Frameworks

• Interactive program guide

• Hardware/software testing
  – by Trask, Roman and Bhanot (2005)
Killer “Killer Example”
Frameworks

- Frameworks are pervasive (e.g. J2EE, Swing, RMI)
- Demonstrate good OO design:
  - inversion of control (user of framework builds components for framework, does not control flow)
  - hotspots (hooks or variability points: variability-commonality analysis or variant-invariant decomposition)
- Presenter Framework: MVC in action
  - provides navigation framework for simple multi-media presentations
  - student can provide content and navigation links using the framework
Killer “Killer Example” Interactive Program Guide

• Example is readily accessible to beginning students
• Rich environment for patterns
  – iterator (channels, programs, themes, etc)
  – state (browse channels, browse themes, set-up, etc)
  – command (behaviors of buttons)
  – mediator (different parts of display must be kept in synch)
Killer “Killer Example”
Hardware / Software Testing

• Addresses the problem of how to build tests for components which don’t yet exist
  – control software for hardware which is being developed in parallel

• Shows progression that developers went through in finding good solution
  – strategy pattern
  – abstract factory pattern
Visit the workshop series website
www.cse.buffalo.edu/~alphonce/KillerExamples

E-mail us:
adrienne@cse.buffalo.edu
mec@daimi.au.dk
alphonce@cse.buffalo.edu