A workshop to share ideas, specifically “killer examples”, on how to teach
Fundamental Object-Oriented (OO) principles and Design Patterns
primarily in an objects-first CS1-CS2 sequence.

**Pedagogical Process of Teaching Design Patterns**

1. Use it
   - 1.1 Use an instance of Iterator to go through a collection.
   - 1.2 General concept of an Iterator.
   - 1.3 Create a class that implements the Iterator interface.
   - 1.4 Separate Iterator to access private parts of a collection.

2. Conceptualize it
   - 2.1 Use an instance of Iterator to go through a collection.
   - 2.2 General concept of Strategy.
   - 2.3.a Create a class that implements Comparator.
   - 2.3.b Create a class hierarchy that implements Strategy.

3. Build it
   - 2.4 GoF presentation of Strategy

4. Analyze/study high quality code
   - High quality code examples

5. Design and construct Software using DPs
   - Use Design Patterns to help test your classes and functions
   - Using Design Patterns to Help Test Your Classes and Functions (Bruce Trask, Angel Roman & Vikram Bhanot)
   - Message-Oriented Middleware Cache Pattern – A Pattern in a SOA Environment (Fang Yan, Ru Fang & Zhung Tran)
   - Using Design Patterns to Help Test Your Classes and Functions (Bruce Trask, Angel Roman & Vikram Bhanot)
   - Message-Oriented Middleware Cache Pattern – A Pattern in a SOA Environment (Fang Yan, Ru Fang & Zhung Tran)

6. Evaluate software
   - Testing Mock-ups (Stubs)
   - Iterative Design Process
e.g.

**Results from this year’s workshop**

**Industrial Examples**

- Using Design Patterns to Help Test Your Classes and Functions
- Message-Oriented Middleware Cache Pattern – A Pattern in a SOA Environment

**Discussion**

- Paul Adamczyk, University of Illinois
- Jürgen Börstler, Umeå University
- Hani Girgis, University of Buffalo, SUNY
- Blake Martin, University of Buffalo, SUNY

**Organizers**

- Carl Alphonce, University of Buffalo, SUNY
- Michael E. Caspersen, University of Aarhus, Denmark
Killer Example

The Jargon File defines a “killer app” as an “application that actually makes a sustaining market for a promising but under-utilized technology.”

In the same vein, we take a “killer example” to be one which provides clear and compelling motivation for some concept.

A View of the Design Process...

Goal of a Killer Example is to show the benefits of using patterns and how these "shortcuts" to solutions eventually break down.

“Killer” idea: Change the requirements to drive students towards higher quality solutions.

Legend

- Students perception of the design process
- True path to desired design
- Problems with design (e.g. smell or errors)
- Student shortcuts to a solution
- Desirable solutions
**Caspersen’s Curve**

**Understanding**

- Can identify proper context
- Can apply DP, given proper context
- Believe DP is good
- Lack of understanding

**Time**

- Motivation
- Use smaller drill/skill program(s) to strengthen understanding
- Reinforcement
- Back in dorm...
- Time for another killer example

**Killer Examples**

Killer Examples must be an integral component of a larger, cohesive OO curriculum.

They do not exist in a vacuum, but rather as part of a deliberate pedagogical progression that drives from motivation to comprehension.

---

**Highlights from previous workshops**

- **Caspersen’s Curve**
- **Killer Example Shown**
- **Back in dorm...**
- **Time for another killer example**

---

**Composition Framework**—D. Skrien (Colby College)
**Configuration Puzzles**—J. Heliotis and S. Marshall (RIT)
**Developing an Elevator Control System**—C. Nevison (Colgate) and B. Wells (South Fork High School)
**Java Power Tools**—R. Rasala, V. K. Proulx and J. Raab (Northeastern)
**Kaleidoscope**—M. R. Wick (University of Wisconsin—Eau Claire)
**Properties of a “Killer Example”**—S. Sendall (Swiss Federal Institute of Technology)

---

**OOPSLA 2004 WORKSHOP**

- **Generic Data Access in Microsoft .NET**—J. Hummel (Lake Forest College)
- **Applying the Extension Object Pattern to the Software Communication Architecture**—D. Paniscotti and B. Trask (SDR Products)
- **Presentation Application (“PowerPoint”)**—S. Stuurman (Open University, The Netherlands) and G. Florijn (SERC)
Motivation for Teaching Design Patterns

We want a systematic way to solve complex problems (need solutions that scale up). Design patterns support the building of correct, robust, flexible & extensible software in an efficient manner (time & $).

Important underlying principles which allow us to reach our goals:

- Abstraction
- Invariant/variant decoupling (commonality/variability analysis)
- Parameterization
- Extreme encapsulation (high abstraction; program to invariant behavior; decoupling to manage complexity)

The underlying principles are applied at different levels:
- method, class, pattern, and framework.