Fourth "Killer Examples" for Design Patterns and

Previous workshops: OOPSLA '02, Seattle •

Results from this year's workshop

Intra Pattern

Inter Patterns

Pedagogical Process of Teaching Design Patterns

Sample patterns:	Iterator	Strategy	...
1. Use it	1.1 Use an instance of Iterator to go through a collection.	2.1 Use an instance of Iterator to go through a collection.	...
2. Conceptualize it	1.2 General concept of an Iterator.	2.2 General concept of Strategy.	...
3. Build it	1.3 Create a class that implements the Iterator interface	2.3.a Create a class that implements Comparator. 2.3.b Create a class hierarchy that implements Strategy.	...
4. Analyze/study high quality code	1.4 Separate Iterator to access private parts of a collection	2.4 GoF presentation of Strategy	...
5. Design and construct Software using DPs	Iterative Design Process e.g. Testing Mock-ups (Stubs)		
6. Evaluate software			

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.

INDUSTRIAL EXAMPLES

Using Design Patterns to Help Test Your Classes and Functions

Bruce Trask, Angel Roman & Vikram Bhanot

PRISMTECH

Message-Oriented Middleware Cache Pattern – A Pattern in a SOA Environment

Fang Yan, Ru Fang & Zhung Tran

IBM China Reseaarch Lab

DISCUS

Paul Adamczyk
University of Illinois

Jürgen Börstler
Umeå University

Hani Girgis
University of Buffalo, SUNY

Blake Martin
University of Buffalo, SUNY

ORGAN

Carl Alphonce

University of Buffalo
SUNY

Michael E. Caspersen

University of Aarhus
Denmark

Objects First Workshop, OOPSLA ’05, San Diego

OOPSLA ’03, Anaheim • OOPSLA ’04, Vancouver

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.

TEACHING EXAMPLES

Design Patterns in JDK Collections	*Killer Lab: Flow Simulation and Lead Poisoning Study*
Christine Bouamalay	**James Heliotis & Carl Lutzer**
SBC Services	Rochester Institute of Technology

SANTS

Jim Caristi
Valparaiso University

Kishore Nair
Earthlink

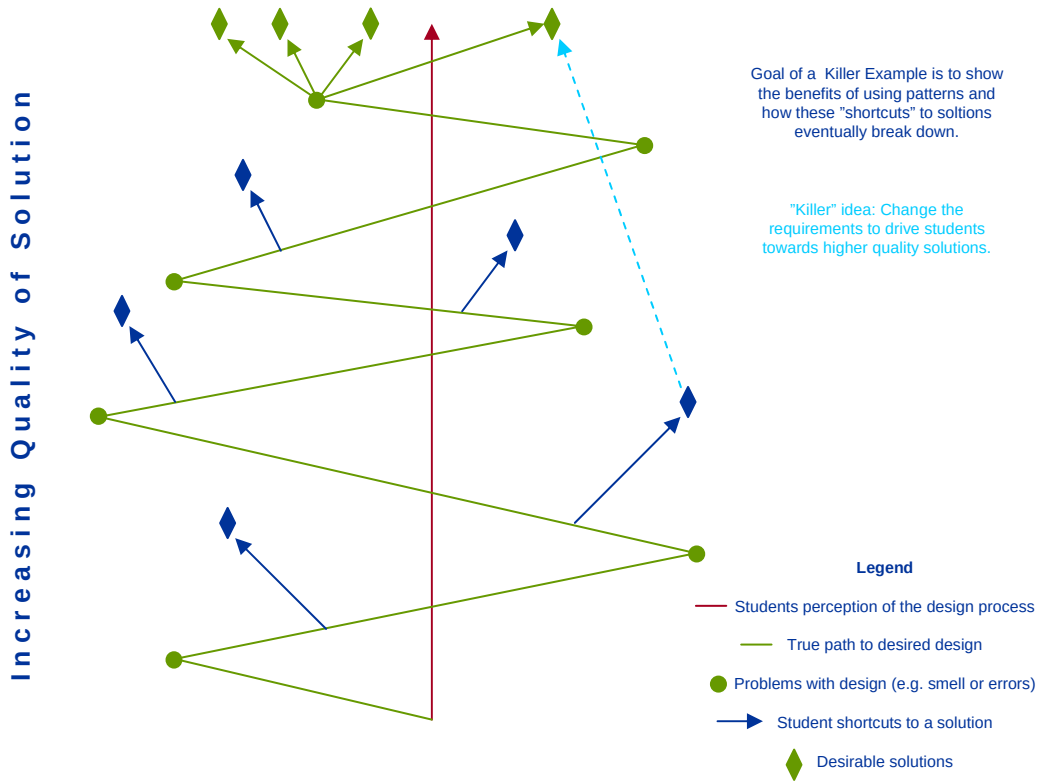
Martha Kosa
Tennessee Technical University

IZERS

Adrianne Decker
University of Buffalo
SUNY

Stephen Wong
Rice University
Houston, Texas

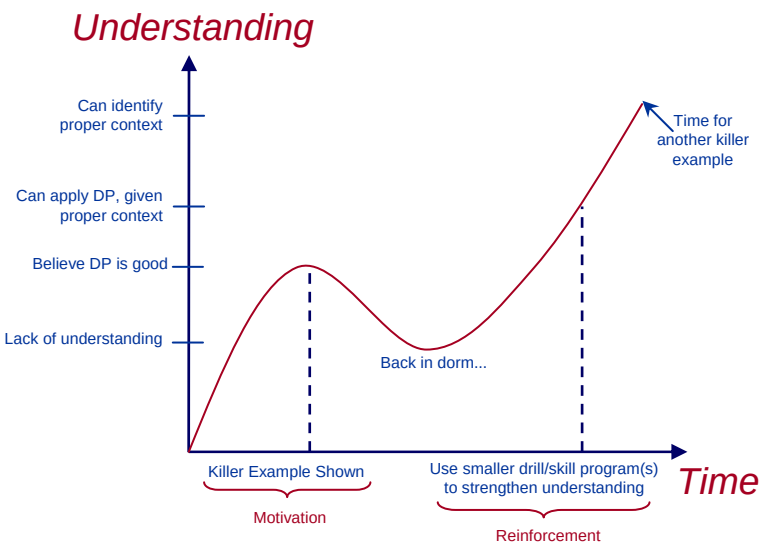
A View of the Design Process…



Results from this year’s workshop

Highlights from previous workshops

Caspersen's Curve



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.

Killer

OOPSLA 2002 Seattle

OOPSLA 2002 WORKSHOP

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'04 VANCOUVER

OOPSLA 2004 WORKSHOP

Generic Data Access in Microsoft .NET—Joe 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)

Examples

OOPSLA 2003 WORKSHOP

A simple calculator for novice learning – J. Bergin (Pace University)
Interactive Program Guide – Asher Sterkin (NDS Technologies)
The Need for Killer Example for Object-Oriented Framworks – M. E. Caspersen and H. B. Christensen (University of Aarhus)
Foundation for Object-Oriented Graphics – R. Rasala (Northeastern University)

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)

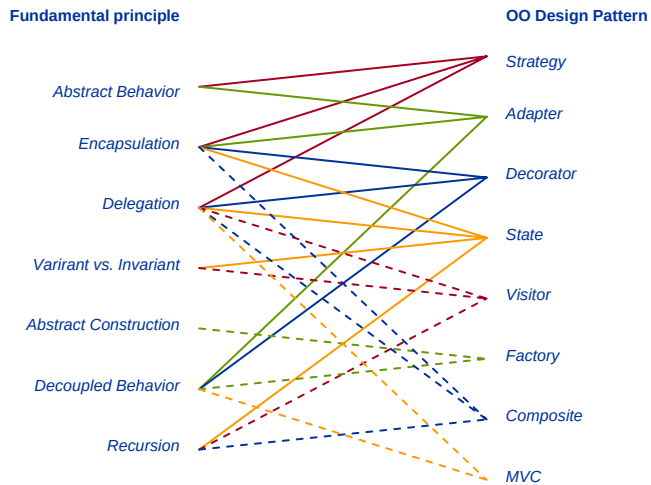
Parameterization

Extreme encapsulation
(high abstraction; program to invariant behavior; decoupliing to manage complexity)

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

Wong's Mapping

Underlying Principles



DPs can be used to illustrate fundamentsl CS principles.

Each DP illustrates multiple principles.
Each principle can be shown with several different DPs.

DP change and shape the way we look at problems.

Highlights from previous workshops