Why Program with Objects?

Why object-orientation?

James Brucker

Problems of Software Development

1. Software is *Complex*.

and the complexity *grows* over time.

Problems of Software Development

2. Software is required to *change*.

Lots of change

- ✓ requirements change
- ✓ our understanding changes
- ✓ technology changes

...and it happens *during* the project

Problems of Software Development

3. Modeling real-world problems is *hard*.

Designing software is *hard.*

Modeling mismatch:

Behavior of real things does not match behavior of software components.

Problems of Software Development

4. Software tends to contain a lot of *defects*.

   This is partially a result of the *complexity*, *change*, *and* *modeling mismatch*.

   It also has to do with our development process

Problems of Software Development

5. Software is expensive to develop and maintain.

Too much *change*, *complexity*, and *mismatch*.

Too many *defects.*

Lack of standard components.

Too much "custom development".

Intrinsic Complexity of Software

"*There is no single development, in either te chnology or in management technique, that by itself promises even one order of magnitude i mprovement in productivity, in reliability, i n simplicity.*"

"*No Silver Bullet*" by Frederick Brooks. *Computer*, 1987

What does he think of O-O Programming? ...

Brooks on O-O

"***Many students of the art hold out more hope for object-oriented programming than for any of the other technical fads of the day.***
I am among them."

"*No Silver Bullet*" by Frederick Brooks. *Computer*, 1987

Read the article to learn <u>why</u> Brooks believes in O-O programming.

Miller’s Law and Complexity

At any one time, a person can concentrate on at most 7 ± 2 chunks (units of information)

- Need to ***limit complexity***.
- How to limit complexity?
  - hide it!
  - modular design
  - only use a module's *public interface*
  - limit dependencies between modules

What We Want

- ✓ **Reduce complexity**... simple interface to parts.
- ✓ **Limit dependency** between parts.
- ✓ Make software components behave **like real things**.
- ✓ Improve **testability**.
- ✓ Enable to **reuse code**.
- ✓ Enable to **reuse entire applications**...

  just "plug in" custom features.

Procedural vs O-O Paradigm

http://www.youtube.com/watch?v=D8jZ0I_GwXQ

Benefit of Object-Orientation (1)

Encapsulate complexity

- divide program into classes
- a class has its own *responsibilities* and *data*
- class has a *well-defined* interface
- *hide* implementation details

Benefit of Object-Orientation (2)

Encapsulate **change**

- a class presents only a simple *public interface*
- *hides* implementation details

as a result...

- we can *localize the effect of change*

Benefit of Object-Orientation (3)

Better abstraction

- objects make good models for things in the real world (problem domain)
- let us think about the problem instead of the code
- *simplify the problem* so we can think about problem without too many details

Benefit of Object-Orientation (4)

Reuse code

- classes are reusable
- *polymorphism* lets us interchange parts
- *inheritance* lets us build new classes that reuse code from old classes.

Reuse components (can be more than 1 class)

- A *facade* makes component look like just one class.
- JavaBeans use this approach.

Benefit of Object-Orientation (5)

Reuse designs

- same situations occur again and again
- *design patterns* for reusable solutions
- *polymorphism and encapsulation* make most of these patterns work

Frameworks - reusable applications

By use of *polymorphism*, an entire application can be reused and customized...

without changing the code.