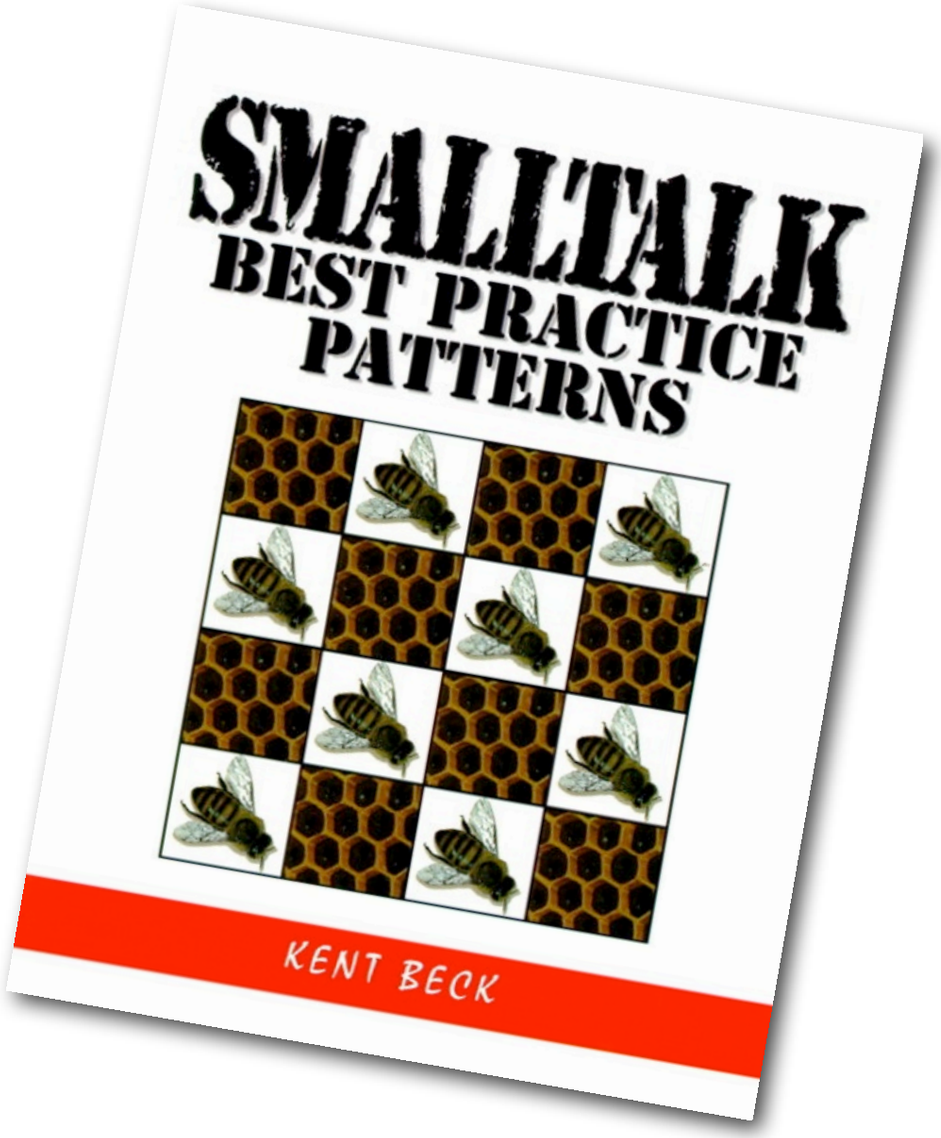
Smalltalk
Best Practice Patterns

Part I

Portland State
UNIVERSITY

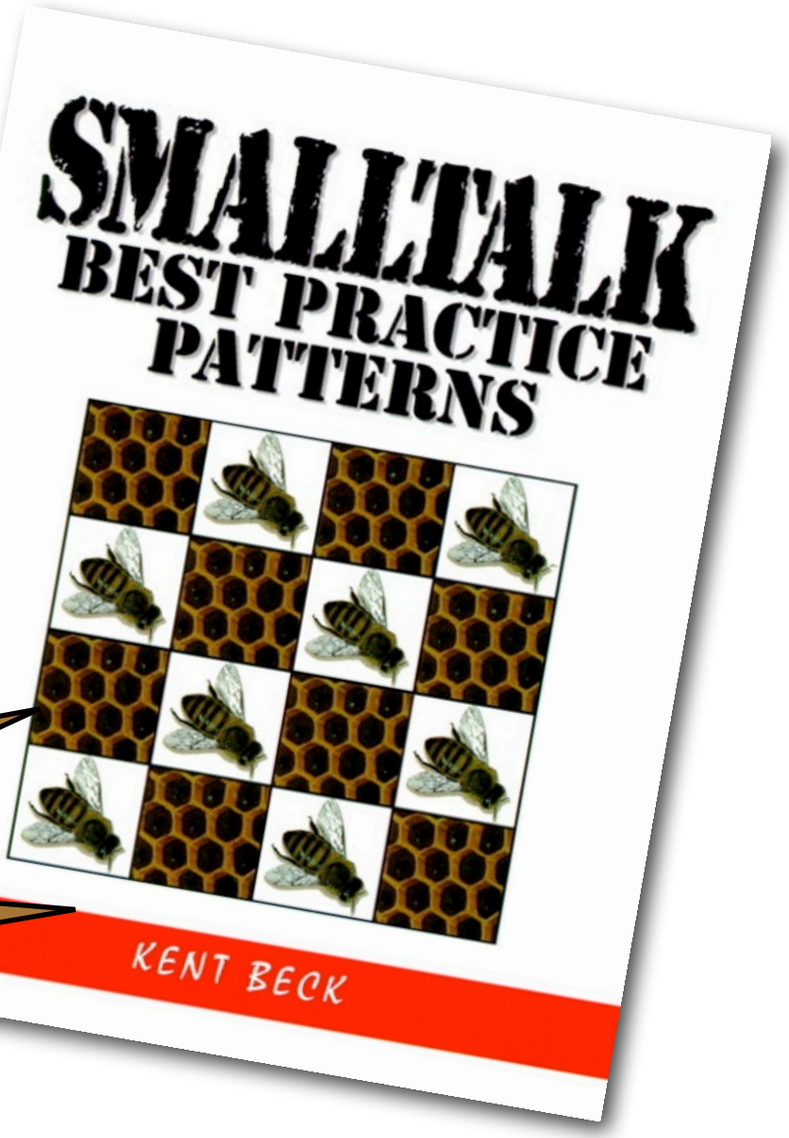
Based on the Book by …

Kent Beck



Based on the Book by …

Kent Beck



Very little here is Smalltalk-specific

Why Patterns?

Why Patterns?

- There are only so many ways of using objects
  - many of the problems that you must solve are independent of the application domain
  - patterns record these problems and successful solutions

Why Patterns?

- There are only so many ways of using objects
  - many of the problems that you must solve are independent of the application domain
  - patterns record these problems and successful solutions
- Remember: the purpose of eduction is to save you from having to think

What's hard about programming?

- Communicating with the computer?
  - not any more!
  - we have made real progress with languages, environments and style
- Communicating with other software developers!
  - 70% of the development budget is spent on "maintenance"
    - discovering the intent of the original programmers

How to improve communication

- Increase bandwidth
  - within the development team
  - between the team and the re-users
- Increase information density
  - say more with fewer bits
  - make our words mean more

A Pattern is:

- A literary form for capturing “best practice”
- A solution to a problem in a context
- A way of packing more meaning into the bytes of our programs

Patterns exist …

- At many levels:
  - Management Patterns
  - Architectural Patterns
  - Design Patterns
  - Programing Patterns
  - Documentation Patterns

Patterns exist …

- At many levels:
  - Management Patterns
  - Architectural Patterns
  - Design Patterns
  - **Programing Patterns**
  - Documentation Patterns

Behavioral Patterns

- *Objects Behave!*
  - Objects contain both state and behavior
  - *Behavior* is what you should focus on getting right!

Patterns for Methods

- Composed Method
- Complete Creation Method
- Constructor Parameter Method
- Shortcut Constructor Method
- Conversion
- Converter Method
- Converter Constructor Method
- Query Method
- Comparing Method
- Execute Around Method
- Debug Printing Method
- Method Comment

Composed Method

How do you divide a program into methods?

- ➡ *Each method should perform one identifiable task*
- ➡ *All operations in the method should be at the same level of abstraction*
- ➡ *You will end up with many small methods*

Complete Creation Method

How do you represent instance creation?

➡ *Don't: expect your clients to use new and then operate on the new object to initialize it.*

➡ *Instead: provide methods that create full-formed instances. Pass all required parameters to them*

- Put creation methods in a protocol called *instance creation*

Non-example:

➡ *Point new x:10; y:20; yourself*

Example:

➡ *Point x:10 y:20*

Constructor Parameter Method

You have a constructor method with parameters. How do you set the instance variables of the new object?

➡ *Define a single method that sets all the variables. Start its name with "set", and follow with the names of the variables*

- Put constructor parameter methods into the *private* protocol
- Answer **self** explicitly (INTERESTING RETURN VALUE)

Why not use the ordinary setter methods?

- *Once and Only Once*
- *Two circumstances:*
  - initialization
  - state-change during computation
- *Two methods*

Shortcut Constructor Methods

What is the external interface for creating a new object when a Constructor Method is too wordy?

➡ *Represent object creation as a method on one of the arguments.*

- Add no more than three such shortcut constructor methods per system!
- Examples: 20@30, key->value,
  20@30 extent: 10@10
- Put shortcut constructor methods into the *converting* protocol

Conversion

How do you convert information from one object's format to another?

- ➡ *Don't: add all possible protocol to every object that may need it*
- ➡ *Instead: convert from one object to another*
  - If you convert to an object with similar responsibilities, use a CONVERTER METHOD.
  - If you convert to an object with different protocol, use a CONVERTER CONSTRUCTOR METHOD

Converter Method

How do you represent simple conversion of another object with the same protocol but a different format?

Kent Beck tells a story …

- *If the source and the destination share the same protocol, and there is only one reasonable way to do the conversion, then provide a method in the source object that converts to the destination.*
- *Name the conversion method "asDestinationClass"*
  - examples: Collection ›› asSet, Number ›› asFloat, but *not* String ›› asDate

Converter Constructor Method

How do you represent the conversion of an object to another with a different protocol?

➡ *Make a constructor method that takes the object-to-be-converted as an argument*

- Put Converter Constructor Methods in the *instance creation* protocol
- Example: Date class ›› fromString:

Query Method

How do you represent the task of testing a property on an object?

What should the method answer?

What should it be named?

➡ *Provide a method that returns a Boolean. Name it by prefacing the property name with a form of "be"—is, was, will, etc.*

Examples:

➡

```
Switch ›› on
        status := #on
Switch ›› off
        status := #off
Switch ›› status
^ status
```

Examples:

➡ *Switch ›› on*
status := #on
Switch ›› off
status := #off
Switch ›› status
^ status

➡ *Switch ›› turnOn*
status := #on
Switch ›› turnOff
status := #off
Switch ›› isOn
^ status = #on
Switch ›› isOff
^ status = #off

Comparing Method

How do you order objects with respect to each other?

- ➡ *Implement <= to answer true if the receiver should be ordered before the argument*
  - Put comparing methods into a protocol called *comparing*
- ➡ *Implement <= only if there is a single overwhelming way to order the objects*