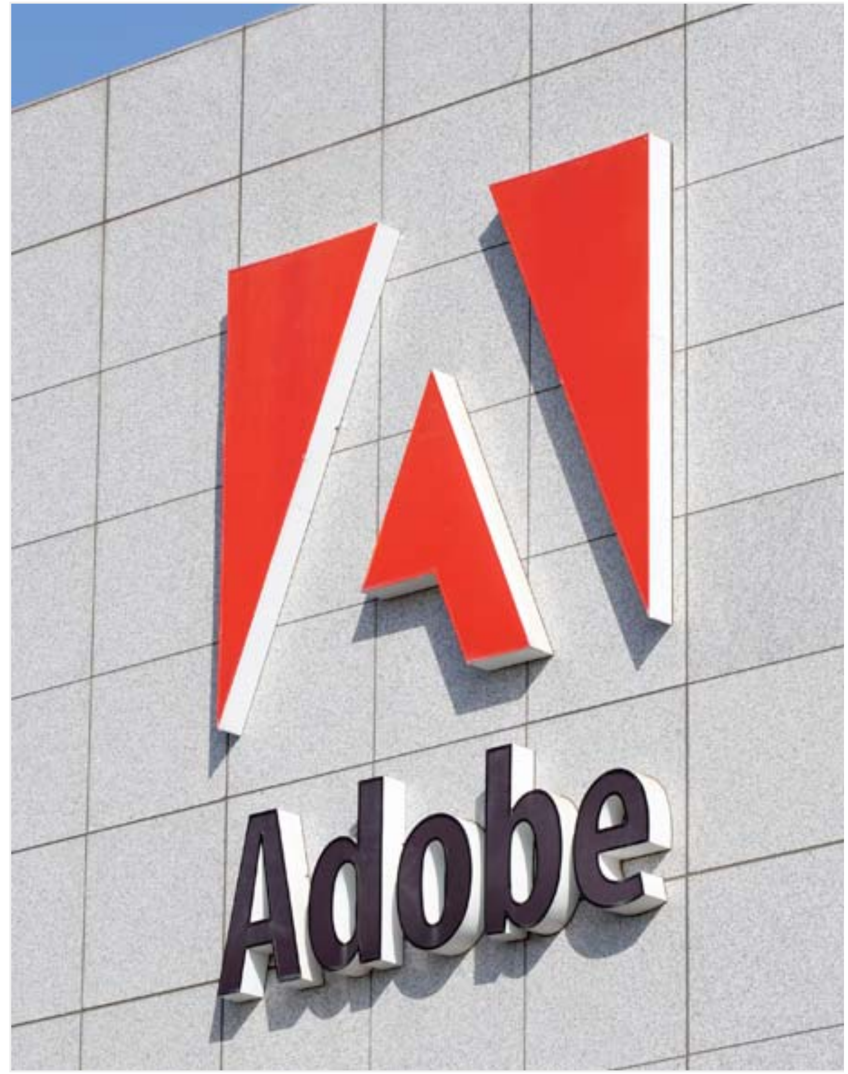
A Possible Future of Software Development

Sean Parent

October 22, 2006



Engineering Team Structure

- Product Line:
  - Photoshop, Acrobat, InDesign, …
- Products:
  - Photoshop CS2, Photoshop Elements, Photoshop Lightroom, …
- Product Team:
  - Developers ≈20
  - Testers ≈30
  - User Interface Designers ≈1
- Shared Technology Groups: ≈20
  - Libraries for Vector Graphics, Type, Color, Help, Localization, XML Parsing, File Handling, etc.

Adobe

Development Process

- Process is Constrained by Business Model
- Schedule Driven, Incremental Development Model on 18-24 month cycles
  - Larger Products and Suites Forced Toward Waterfall Model
  - Press for Manuals must be reserved up to 5 months in advance
- Most Products Ship Simultaneously For Macintosh and Windows in English, French, German, and Japanese
  - Other languages follow shortly to about 24 languages

Photoshop Facts

- History
  - 1987: Started by Thomas Knoll
  - 1990: 1.0 Shipped by Adobe
  - **1991: 2.0 Clipping Path**
  - 1993: 2.5 First Version on Windows
  - 1994: 3.0 Layers
  - 1996: 4.0 Actions & Adjustment Layers
  - 1998: 5.0 History & Color Management
  - **1999: 5.5 Web Development**
  - 2000: 6.0 Typography
  - **2002: 7.0 Camera RAW, Healing Brush, Natural Painting**
  - 2003: CS Lens Blur, Color Match, Shadow/Highlight
  - 2005: CS2 High Dynamic Range Imaging, Smart Objects, Lens Correction

Adobe

Photoshop Code

- 100% C++ since Photoshop 2.5
- Stats for Photoshop CS2 (version 9):
  - Files: ≈ 6,000
  - Lines: ≈ 3,000,000
  - Developers: 20
  - Testers: 28
- Develop Cycle: ≈18 months
- Image Processing Code: ≈15%

The Analysts Future

- "**Best practices**", methodologies, and process are changing continuously
- Trend towards **Java** and C# languages
  - As well as JavaScript and VisualBasic is still strong
  - Java still has only a small presence on the desktop
  - Object Oriented is Ubiquitous
- **XML** growing as Data Interchange Format
- Web Services
- Open Source
  - Foundation Technologies Commoditized

The Analysts Future

- "Organizations need to integrate security best practices, security testing tools and security-focused processes into their software development life cycle. Proper execution improves application security, reduces overall costs, increases customer satisfaction and yields a more-efficient SDLC."
  - Gartner Research, Feburary 2006

- "Microsoft has been slowly moving to a new development process that will affect how partners and customers evaluate and test its software… The new process should help Microsoft gain more feedback earlier in the development cycle, but it won't necessarily help the company ship its products on time or with fewer bugs."
  - Directions on Microsoft, March 2006

Adobe

Why Status Quo Will Fail

- "I've assigned this problem [binary search] in courses at Bell Labs and IBM. Professional programmers had a couple of hours to convert the description into a programming language of their choice; a high-level pseudo code was fine… Ninety percent of the programmers found bugs in their programs (and I wasn't always convinced of the correctness of the code in which no bugs were found)."
  - Jon Bentley, Programming Pearls, 1986

Adobe

Binary Search Solution

```
int* lower_bound(int* first, int* last, int x)
{
    while (first != last)
    {
        int* middle = first + (last - first) / 2;

        if (*middle < x) first = middle + 1;
        else last = middle;
    }

    return first;
}
```

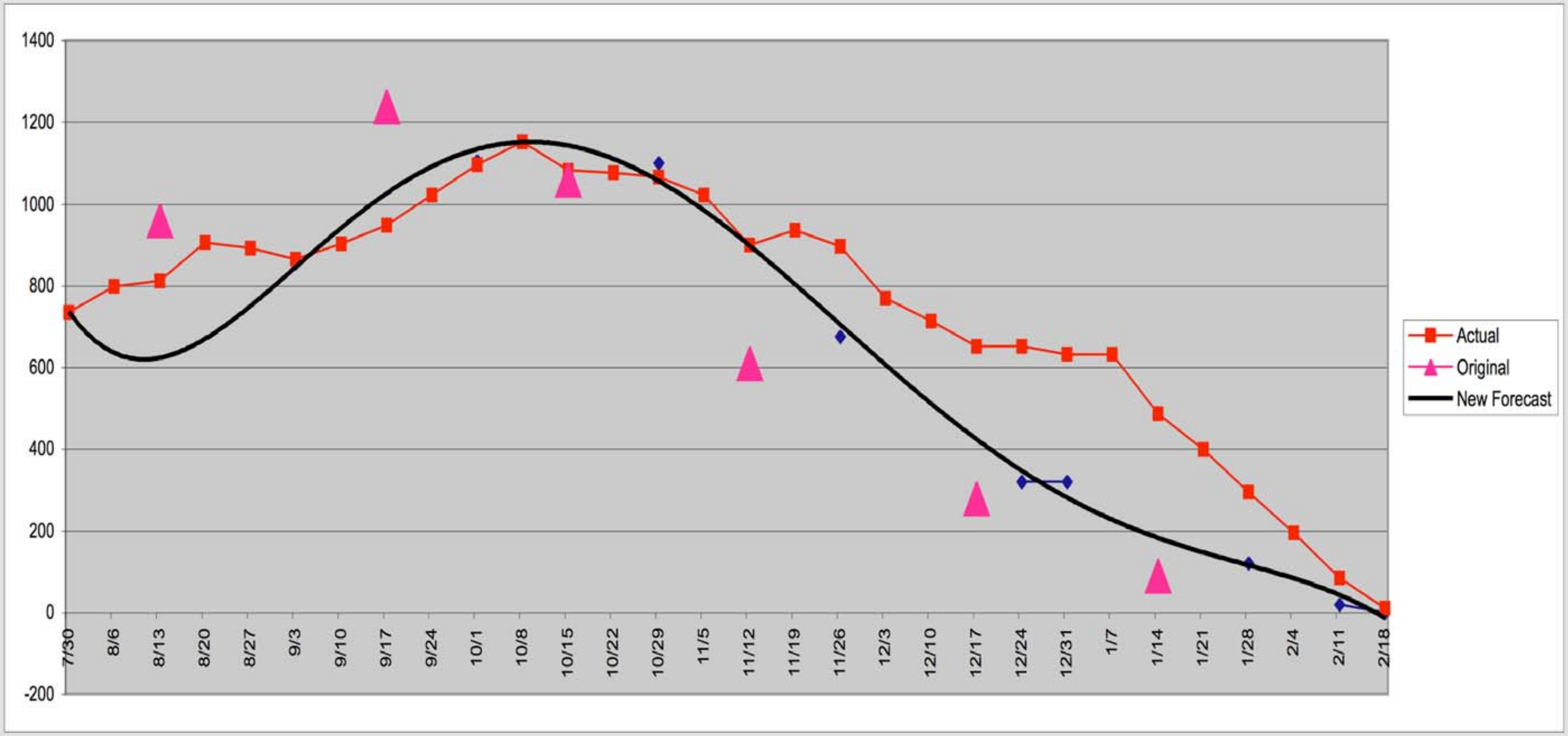

Adobe

Question: If We Can't Write Binary Search…

- Jon Bentley's solution is considerably more complicated (and slower).
- Photoshop uses this problem as a take home test for candidates.
  - **More than 90% of candidates fail**.
- Our experience teaching algorithms would indicate that more than 90% of engineers, regardless of experience, cannot write this simple code.

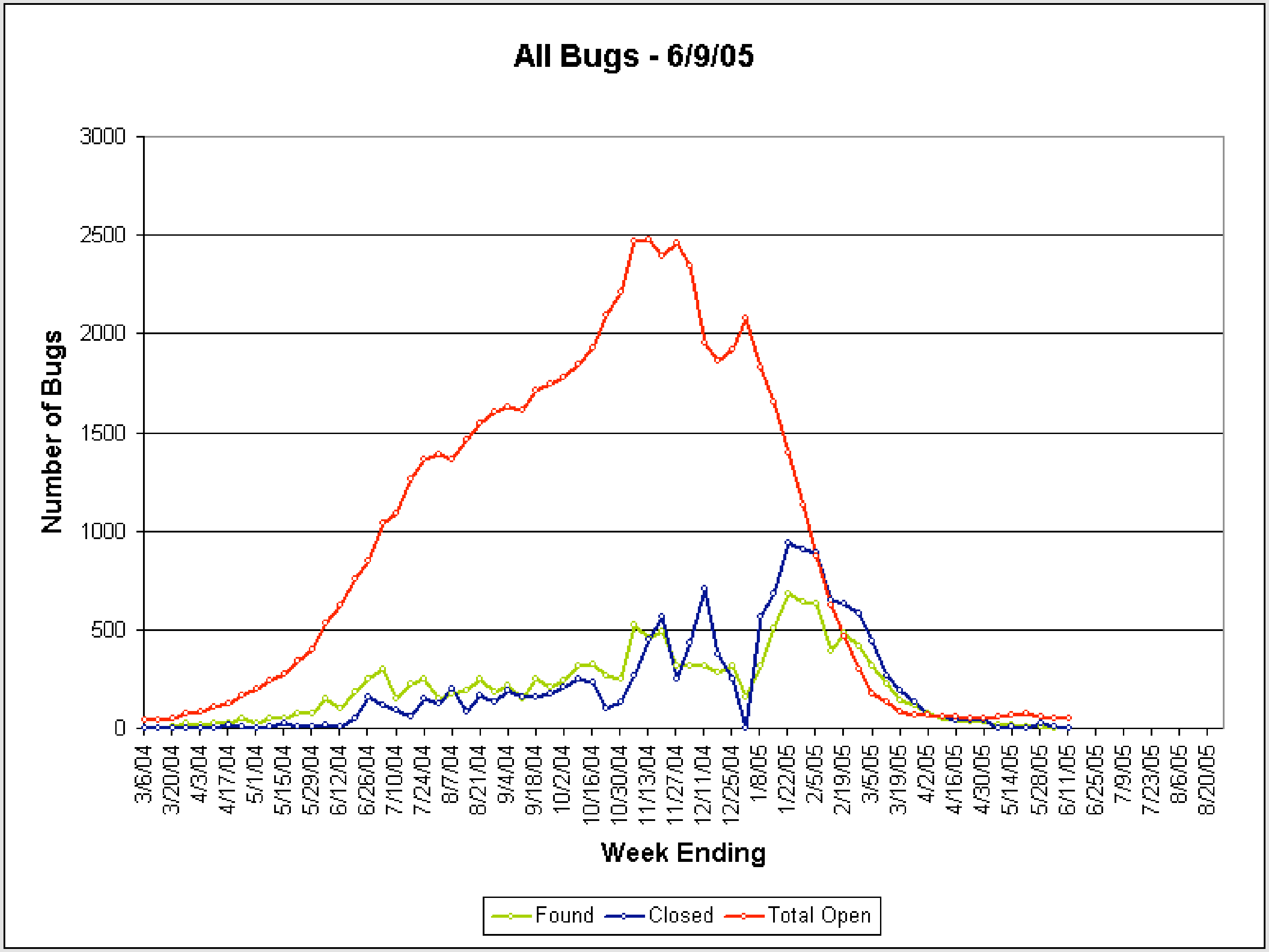
- …then how is it possible that Photoshop, Acrobat, and Microsoft Word exist?

Bugs During Product Cycle



Adobe

Bugs During Product Cycle



Answer: Iterative Refinement.

- Current programming methodologies lend themselves to iterative refinement.
- We don't solve problems, we approximate solutions.

Writing Correct Algorithms

- We need to study how to write correct algorithms.
- Write algorithms once in a general form that can be reused.
- Focus on the common algorithms actually used.

Generic Programming

- Start with a concrete algorithm.
- Refine the algorithm, reducing it to its minimal requirements.
- Clusters of related requirements are known as Concepts.
- Define the algorithms in terms of Concepts - supporting maximum reuse.
- Data structures (containers) are created to support algorithms.

Programming as Mathematics

- Generic Programming
  - Semantic Requirement
  - Concept
  - Model (types model concepts)
  - Algorithms
  - Regular Function
  - Complexity

- Mathematics
  - Axiom
  - Algebraic Structure
  - Model
  - Theorems
  - Function
  - ___________

- Refined Concept - a Concept defined by adding requirements to an existing concept.
  - monoid: semigroup with an identity element
  - BidirectionalIterator: ForwardIterator with constant complexity decrement
- Refined Algorithm - an algorithm performing the same function as another but with lower complexity or space requirements on a refined concept

Adobe

Simple Generic Algorithm

```
template <typename T> // T models Regular
void swap(T& x, T& y)
{
    T tmp(x);
    x = y;
    y = tmp;
}
```


Adobe

A Quick Look At Concepts

expression	return type	post-condition
T(t)		t is equal to T(t)
T(u)		u is equal to T(u)
t.~T()		
&t	T*	denotes address of t
&u	const T*	denotes address of u

Table 1 - CopyConstructable

t = u	T&	t is equal to u

Table 2 - Assignable

a == b	bool	== is the equality relation

Table 3 – EqualityComparable

Value Semantics

- For all a, a == a (reflexive).
- If a == b, then b == a (symmetric).
- If a == b, and b == c, then a == c (transitive).
- !(a == b) ⇔ a != b.
- T a(b) implies a == b.
- T a; a = b ⇔ T a(b).
- T a(c); T b(c); a = d; then b == c.
- T a(c); T b(c); modify(a) then b == c and a != b.
- If a == b then for any regular function f, f(a) == f(b).

Challenges

- Language Support for Concepts
- Extending Concepts to Runtime
  - Replacing inheritance as a mechanism for polymorphism
- Constructing a Library of Algorithms and Containers
  - STL is only a beginning - must be considered an example

Question: Is this enough to build an application?

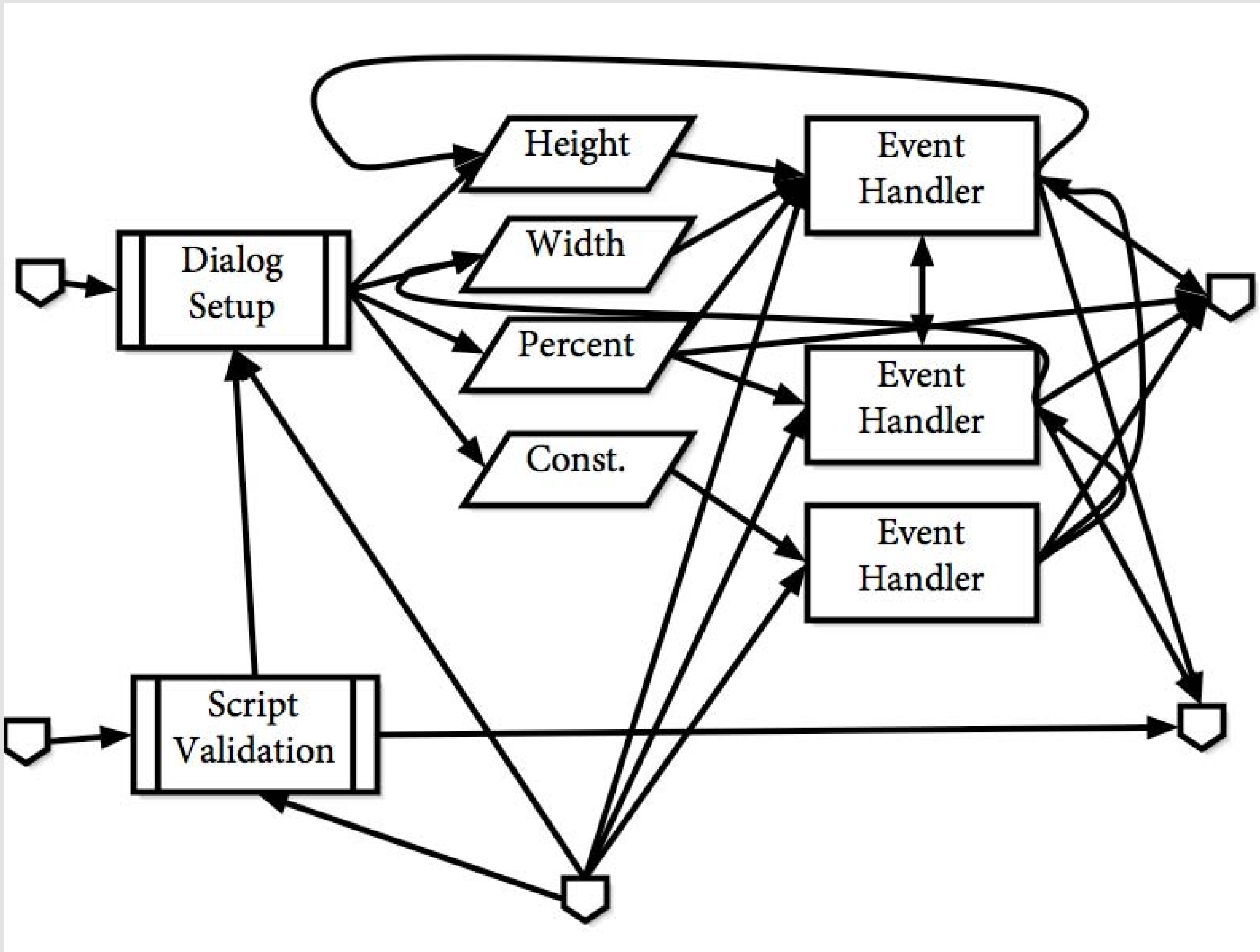
Better by Adobe.™

Adobe

Current Design of Large Systems

- Networks of objects form implicit data structures.
- Messaging among objects form implicit algorithms.
- Design Patterns assist in reasoning about these systems.
  - Local rules which approximate correct algorithms and structures.
- Iteratively refine until quality is “good enough.”

Event Flow in a Simple User Interface



Facts:

- 1/3 of the code in Adobe's desktop applications is devoted to event handling logic.
- 1/2 of the bugs reported during a product cycle exist in this code.

If Writing Correct Algorithms is Difficult…

- …Writing correct implicit algorithms is very difficult.
- We need to study what these implicit algorithms do, and express the algorithms explicitly on declared data structures.

Declarative Programming

- Describe software in terms of rules rather than sequences of instructions.
  - Rules define a structure upon which solving algorithms can operate.
- Examples of non-Turing complete* systems:
  - Lex and YACC (and BNF based parsers)
  - Sequel Query Language (SQL)
  - HTML (if we ignore scripting extensions)
  - Spreadsheet
- Can be Turing complete (i.e. Prolog).
  - But Turing complete systems lead us back to the complexity of algorithms.

**Some of these systems are "accidentally" Turing complete or support extensions that make them Turing complete (such as allowing cycles in a spreadsheet engine). In practice though, this can often be effectively ignored and disallowed.*

STLab Research: “Declarative UI Logic”

- Definition: A User Interface (UI) is a system for assisting a user in selecting a function and providing a valid set of parameters to the function.
- Definition: A Graphical User Interface (GUI) is a visual and interactive system for assisting a user in selecting a function and providing a valid set of parameters to the function.
- We’re starting with what it means to assist the user in providing a valid set of parameters to a function…

Demo

Imperative Solution to Mini-Image Size

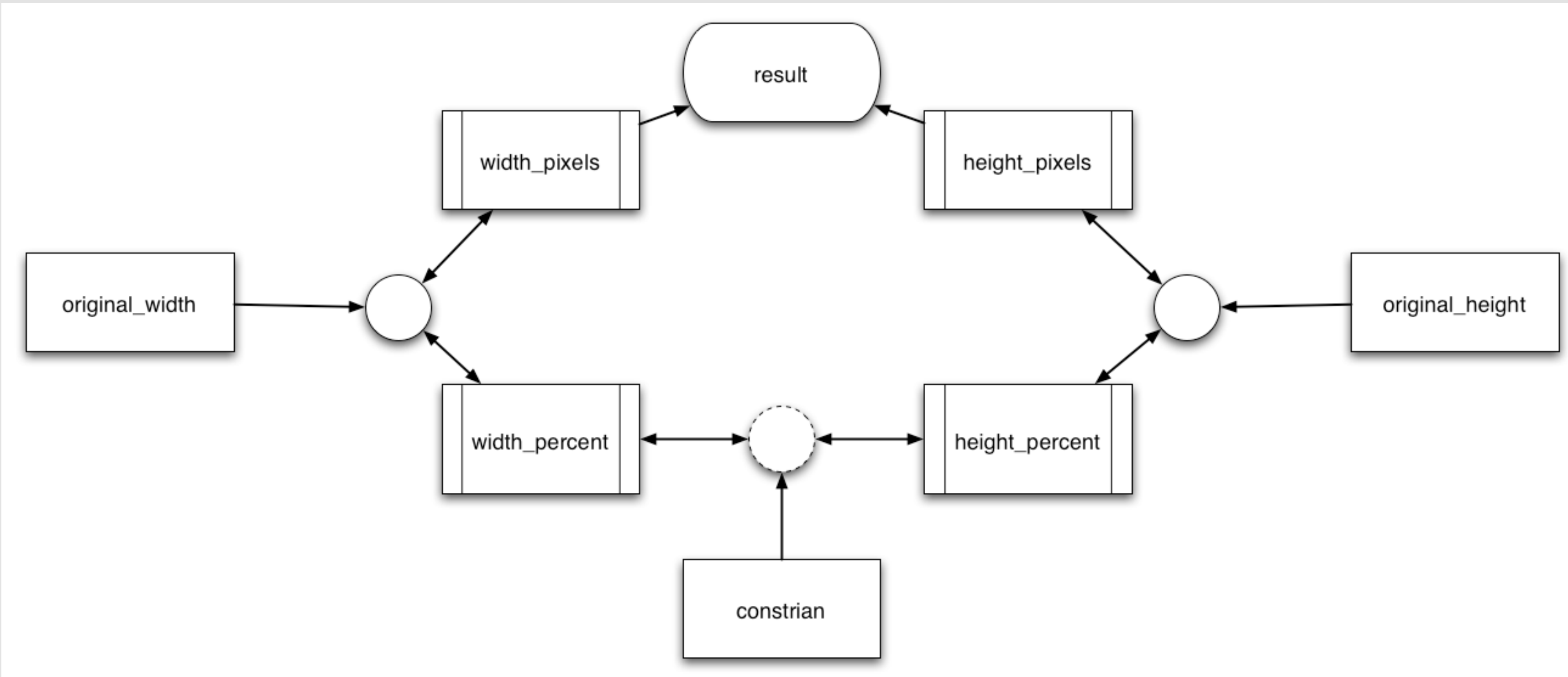
```objc
#import "ImageSizeController.h"
#import <Foundation/NSObject.h>
#import <AppKit/NSNibDeclarations.h>
#import <AppKit/NSControl.h>
#import <AppKit/NSCell.h>
#import <Foundation/NSNumberFormatter.h>
#import <Foundation/NSNotification.h>
#import <AppKit/NSTextField.h>
#import <math.h>
#import <stddef.h>

/* Reading a text field with a formatter attached forces
the text through the formatter. */

static double TextField_unformattedDoubleValue(
                id textField ) {
    id formatter = [ textField formatter ];
    [ textField setFormatter: nil ];
    double result = [ textField doubleValue ];
    [ textField setFormatter: formatter ];
    return result;
    }


/* Same logic but for integers. */

static double TextField_unformattedIntValue(
                id textField ) {
    id formatter = [ textField formatter ];
    [ textField setFormatter: nil ];
    int result = [ textField intValue ];
    [ textField setFormatter: formatter ];
    return result;
    }


/* Setting a text field while it is editing doesn't manage
to set the text. So, we have to stop editing and the
restart. */

static void TextField_setDoubleValueAndFormatter(
                id textField,
                double value,
                NSFormatter *formatter ) {
    BOOL wasEditing = [ textField abortEditing ];

    /* If we're changing the formatter, then we want to
    make sure that the display updates including the edit
    field. */

    if( [ textField formatter ] != formatter ) {
        [ textField setFormatter: formatter ];
        [ textField setDoubleValue: value - 1.0 ];
    }

    [ textField setDoubleValue: value ];

    if( wasEditing ) {
        [ textField selectText: nil ];
    }
}


/* Same logic but with integer values. */

static void TextField_setIntValueAndFormatter(
                id textField,
                int value,
                NSFormatter *formatter ) {
    BOOL wasEditing = [ textField abortEditing ];

    /* If we're changing the formatter, then we want to
    make sure that the display updates including the edit
    field. */

    if( [ textField formatter ] != formatter ) {
        [ textField setFormatter: formatter ];
        [ textField setIntValue: value - 1 ];
    }

    [ textField setIntValue: value ];

    if( wasEditing ) {
        [ textField selectText: nil ];
    }
}

/* Here is the class declaration for the controller. */

@interface ImageSizeController : NSObject {
    IBOutlet id heightField_;
    IBOutlet id widthField_;
    IBOutlet id constrainProportionsBox_;
    IBOutlet id usePercentagesBox_;
    IBOutlet NSNumberFormatter *pixelFormatter_;
    IBOutlet NSNumberFormatter *percentFormatter_;
@private
    int initialWidthPixels_;
    int initialHeightPixels_;
    int widthPixels_;
    int heightPixels_;
    double widthPercentage_;
    double heightPercentage_;
    BOOL constrainProportions_;
    BOOL usePercentages_;
}
- (void) showWidth;
- (void) showHeight;
- (void) showAll;
- (IBAction) heightAction: (id)sender;
- (IBAction) widthAction: (id)sender;
- (IBAction) constrainProportionsAction: (id)sender;
- (IBAction) usePercentagesAction: (id)sender;
- (IBAction) apply: (id)sender;
- (IBAction) revert: (id)sender;
- (void) awakeFromNib;
@end


@implementation ImageSizeController

/* Update the width field. */

- (void) showWidth {
    if( usePercentages_ ) {
        TextField_setDoubleValueAndFormatter(
            widthField_, widthPercentage_,
            percentFormatter_ );
    } else {
        TextField_setIntValueAndFormatter(
            widthField_, widthPixels_, pixelFormatter_ );
    }
}


/* Update the height field. */

- (void) showHeight {
    if( usePercentages_ ) {
        TextField_setDoubleValueAndFormatter(
            heightField_, heightPercentage_,
            percentFormatter_ );
    } else {
        TextField_setIntValueAndFormatter(
            heightField_, heightPixels_, pixelFormatter_ );
    }
}


/* Update width and height fields. */

- (void) showWidthAndHeight {
    [ self showWidth ];
    [ self showHeight ];
}


/* Update all controls. */

- (void) showAll {
    [ self showWidthAndHeight ];
    [ usePercentagesBox_ setState:
          usePercentages_ ? NSOnState : NSOffState ];
    [ constrainProportionsBox_ setState:
          constrainProportions_ ? NSOnState : NSOffState ];
}


/* Revert the width and height. This works regardless of
the checkbox states. */

- (void) revertWidthAndHeight {
    widthPixels_ = initialWidthPixels_;
    widthPercentage_ = 100.0;

    heightPixels_ = initialHeightPixels_;
    heightPercentage_ = 100.0;

    [ self showWidthAndHeight ];
}


/* The revert button does its work via
revertWidthAndHeight. */

- (IBAction) revert: (id) sender {
    [ self revertWidthAndHeight ];
}


/* Handle the apply button by copying over the width and
height. This also sets the percentage values. If we are
displaying percentages, then we need to update. We update
for pixels as well in case this forced any rounding. */

- (IBAction) apply: (id) sender {
    initialWidthPixels_ = widthPixels_;
    widthPercentage_ = 100.0;

    initialHeightPixels_ = heightPixels_;
    heightPercentage_ = 100.0;

    [ self showWidthAndHeight ];
}


/* Handle an event from the use percentages checkbox. */

- (IBAction) usePercentagesAction: (id) sender {
    BOOL newUsePercentages = [ sender state ] == NSOnState;
    if( newUsePercentages != usePercentages_ ) {
        usePercentages_ = newUsePercentages;
        [ self showWidthAndHeight ];
    }
}


/* Handle an event from the constrain proportions checkbox.
*/

- (IBAction) constrainProportionsAction: (id) sender {
    BOOL newConstrainProportions =
            [ sender state ] == NSOnState;
    if( newConstrainProportions != constrainProportions_ ){
        constrainProportions_ = newConstrainProportions;
        if( newConstrainProportions ) {
            [ self revertWidthAndHeight ];
        }
    }
}


/* The following routines handle conversion between pixels
and percentages for width and height. */

- (void) widthPixelsFromPercentage {
    widthPixels_ = (int)
        floor( initialWidthPixels_ * widthPercentage_
                / 100.0 + 0.5 );
}


- (void) widthPercentageFromPixels {
    widthPercentage_ =
        widthPixels_ * 100.0 / initialWidthPixels_;
}


- (void) heightPixelsFromPercentage {
    heightPixels_ = (int)
        floor( initialHeightPixels_ * heightPercentage_
                / 100.0 + 0.5 );
}


- (void) heightPercentageFromPixels {
    heightPercentage_ =
        heightPixels_ * 100.0 / initialHeightPixels_;
}

/* Process a change to the width field. */

- (IBAction) widthAction: (id) sender {
    if( usePercentages_ ) {
        widthPercentage_ =
            TextField_unformattedDoubleValue( sender );
        [ self widthPixelsFromPercentage ];
    } else {
        widthPixels_ =
            TextField_unformattedIntValue( sender );
        [ self widthPercentageFromPixels ];
    }

    if( constrainProportions_ ) {
        heightPercentage_ = widthPercentage_;
        [ self heightPixelsFromPercentage ];
        [ self showHeight ];
    }
}


/* Process a change to the height field. */

- (IBAction) heightAction: (id) sender {
    if( usePercentages_ ) {
        heightPercentage_ =
            TextField_unformattedDoubleValue( sender );
        [ self heightPixelsFromPercentage ];
    } else {
        heightPixels_ =
            TextField_unformattedIntValue( sender );
        [ self heightPercentageFromPixels ];
    }

    if( constrainProportions_ ) {
        widthPercentage_ = heightPercentage_;
        [ self widthPixelsFromPercentage ];
        [ self showWidth ];
    }
}

/* Trigger the text field actions in response to actual
changes. */

    - (void) controlTextDidChange:
    -           (NSNotification *) notification {
    id sender = [ notification object ];
    SEL action = [ sender action ];
    if( action ) {
        [ [ sender target ]
            performSelector: action
            withObject: sender ];
    }
}

/* When we start up, we want to set initial values. This
would ordinarily be
done by code that was creating the controller and then
running it with the dialog
NIB, but we aren't worrying about that here. */

- (void) awakeFromNib {
    initialWidthPixels_ = widthPixels_ = 400;
    initialHeightPixels_ = heightPixels_ = 300;
    widthPercentage_ = 100.0;
    heightPercentage_ = 100.0;
    constrainProportions_ = YES;
    usePercentages_ = NO;
    [ self showAll ];
}

@end
```


Adobe

Declarative Solution

```
sheet mini_image_size
{
 input:
    original_width      : 5 * 300;
    original_height     : 7 * 300;
 interface:
    constrain           : true;
    width_pixels        : original_width     <== round(width_pixels);
    height_pixels       : original_height    <== round(height_pixels);
    width_percent;
    height_percent;
 logic:
    relate {
        width_pixels        <== round(width_percent * original_width / 100);
        width_percent       <== width_pixels * 100 / original_width;
    }
    relate {
        height_pixels       <== round(height_percent * original_height / 100);
        height_percent      <== height_pixels * 100 / original_height;
    }
    when (constrain) relate {
        width_percent       <== height_percent;
        height_percent      <== width_percent;
    }
 output:
    result <== { height: height_pixels, width: width_pixels };
}
```

Structure of Simple User Interface



Future of Software Development

- 85% of existing code base can be replaced with small declarations and a small library of generic algorithms.
- Formally describe application behavior by expressing algorithm requirements and structure invariants.
- Extend the ideas from STL to encompass richer structures with full transaction semantics.
- Shift polymorphic requirements from objects to containers allowing generic programming with runtime polymorphism.

Adobe

Better by Adobe.™