C++ now

2025

Keynote: Are We There Yet?

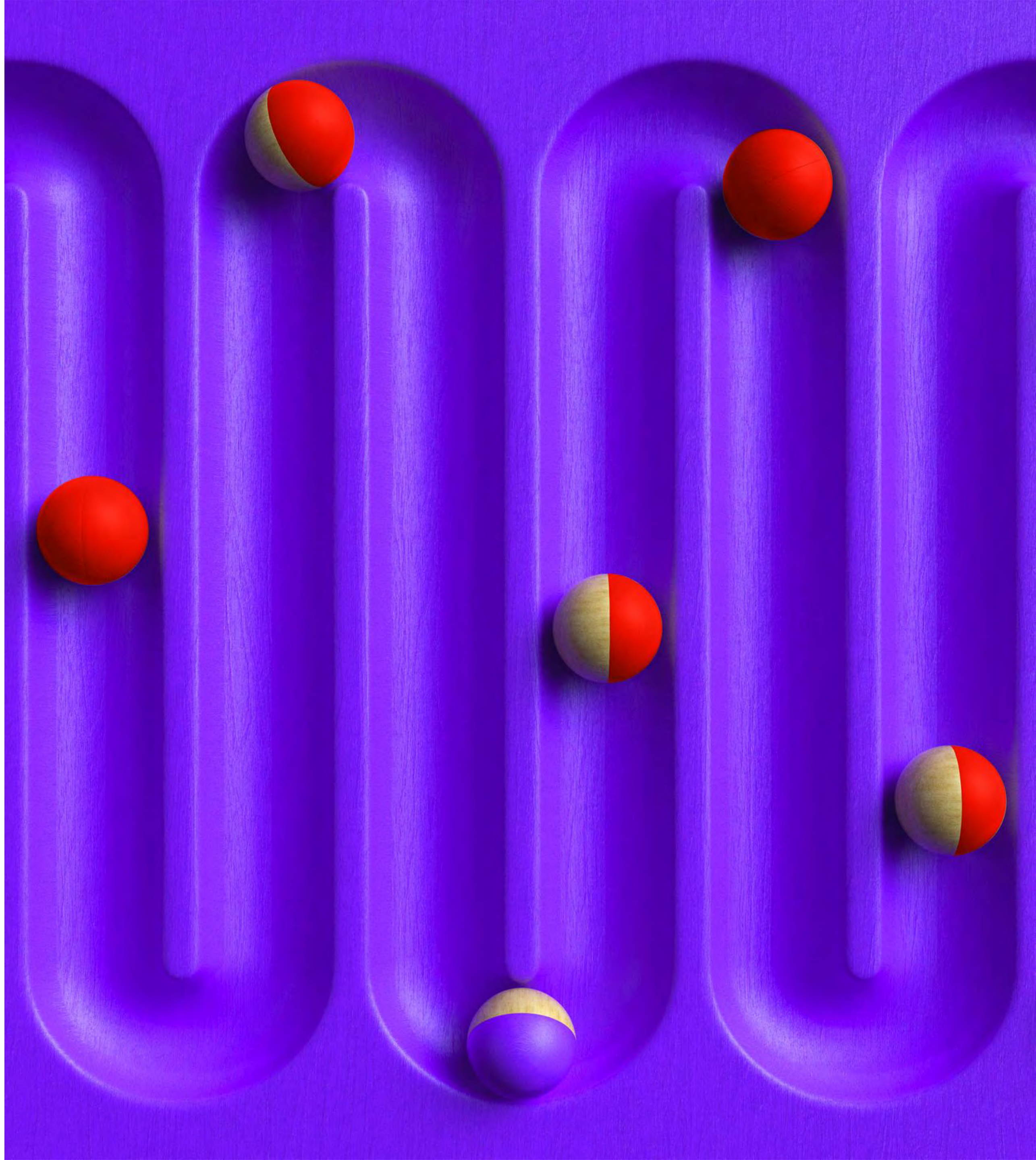
Sean Parent

Are We There Yet?

Sean Parent | Sr. Principal Scientist
Software Technology Lab

Adobe

Image by Bruno Tornielli



Industry Developments

Adobe

	2006	2025
Employees	6,000	30,000
Revenue	$2.6B	$22.0B

Photoshop

	2006	2025
Engineers	20	250
Quality Engineers	30	60
Platforms	macOS, Windows	macOS, Windows, Linux (server), iOS, iPadOS, Browser (WASM), …
Shared Technology Groups	✓	✓
Process	Waterfall, 18-24 month cycles	Agile(ish)

Analysts (Then)

“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, February 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

Why the Status Quo Will Fail (2006)

"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

Jon Bentley’s Solution (translated to C++)

```
int binary_search(int x[], int n, int v) {
    int l = 0;
    int u = n - 1;

    while (true) {
        if (l > u) return -1;

        int m = (l + u) / 2;

        if (x[m] < v) l = m + 1;
        else if (x[m] == v) return m;
        else /* (x[m] > v) */ u = m - 1;
    }
}
```

Binary Search Solution (from 2006 talk)

```
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;
}
```

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?

Analysts (Now)

“Almost all developers will increase their use of generative AI and machine learning as code assistants and low-code/no-code tools proliferate.”

– IDC Modern Software Development Survey, September 2024

“More than **55% of developers** automatically generate **40% or more of their code** in their IDEs, highlighting the widespread use of generative AI and machine learning in software development.”

– IDC Modern Software Development Survey, September 2024

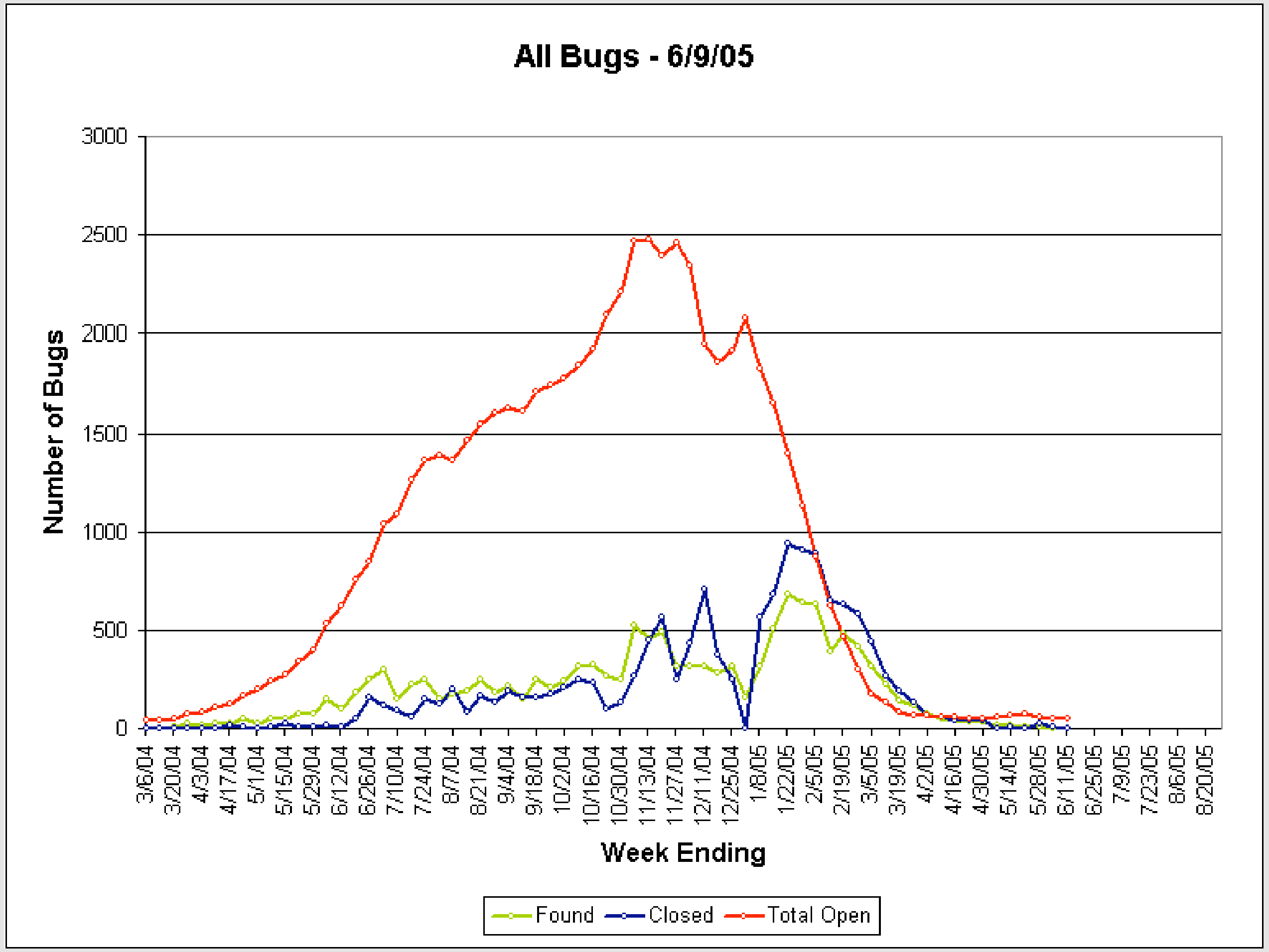
Binary Search Solution (Copilot 2025)

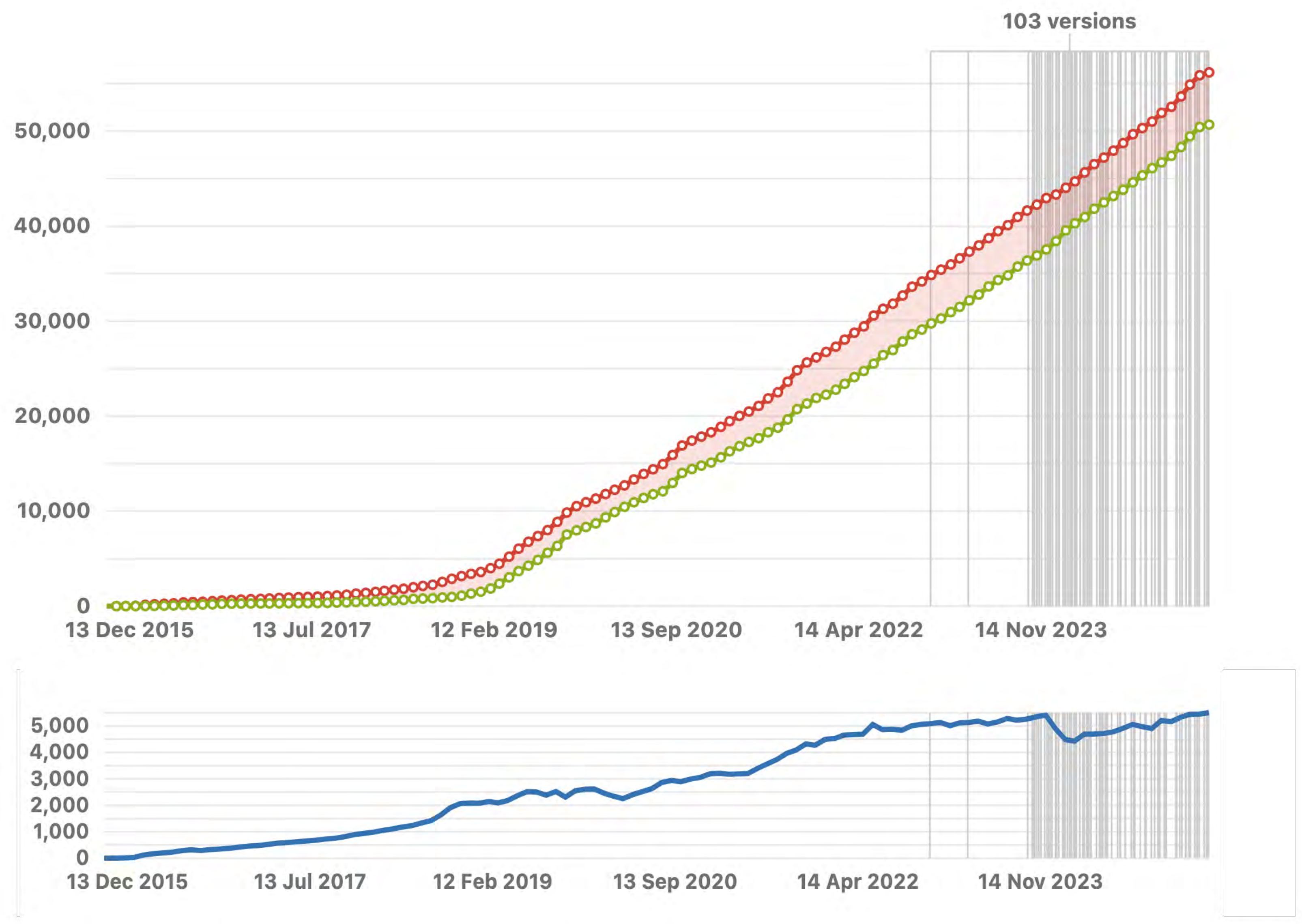
```
int* binary_search_insert_position(int* first, int* last, int x) {
    while (first < last) {
        int* mid = first + (last - first) / 2;
        if (*mid < x) {
            first = mid + 1;
        } else {
            last = mid;
        }
    }
    return first;
}
```

- “**Algorithm Naming:** The function is essentially an adaptation of std::lower_bound().”

Adobe

Bugs During Product Cycle

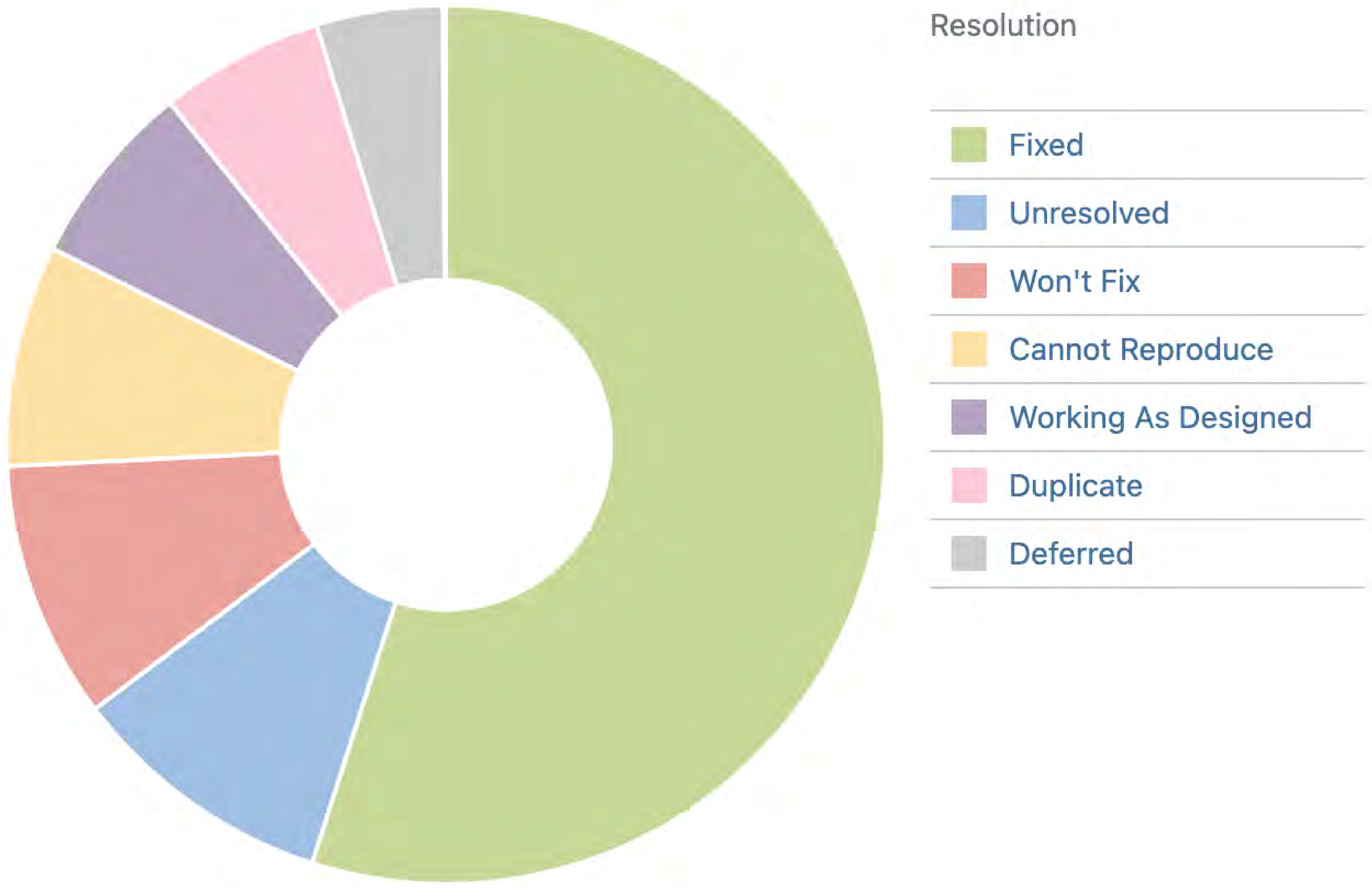


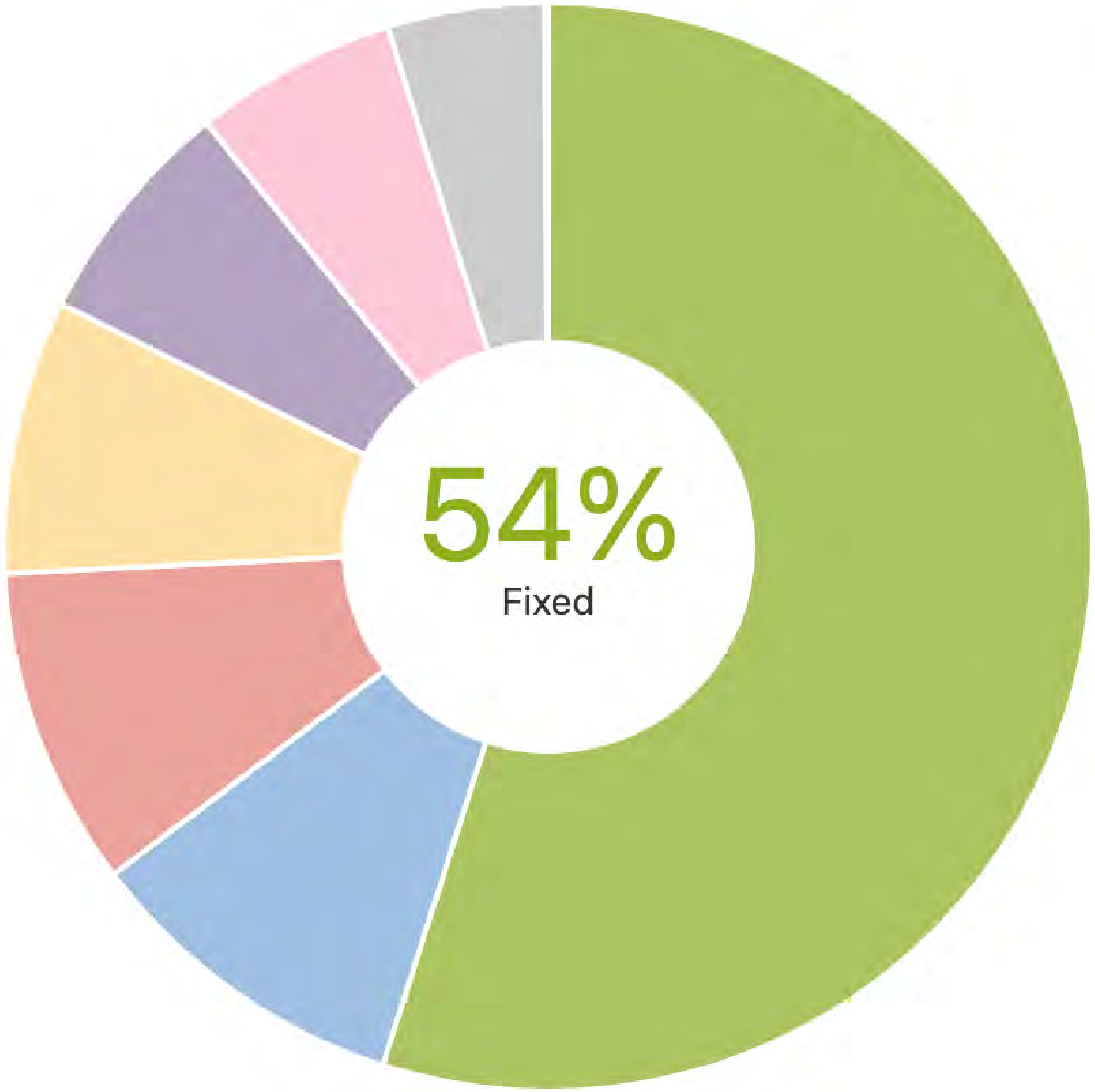


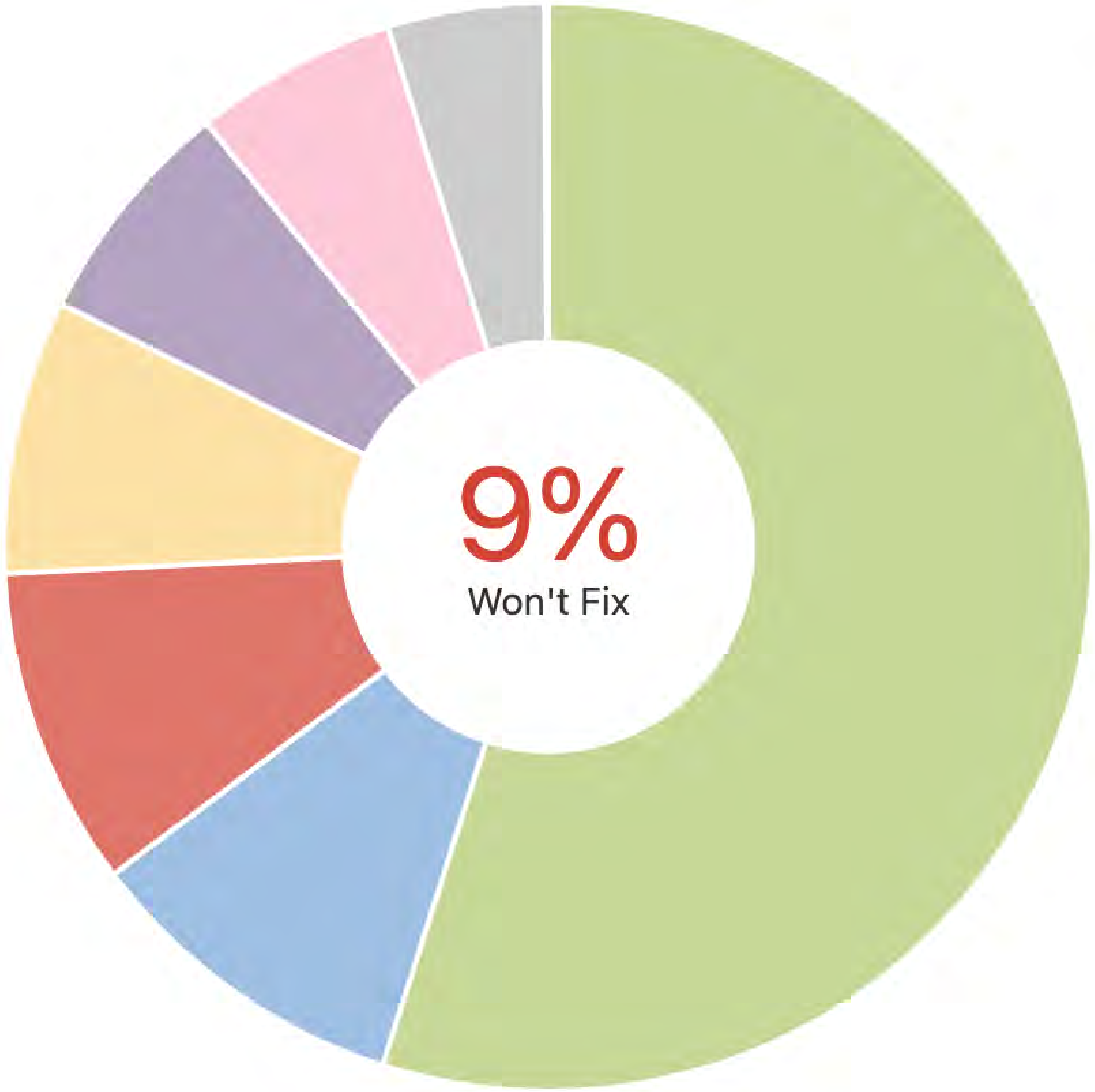
Issues in the last 3,476 days (grouped monthly) View in Issue Navigator

○ Created issues (56180)

○ Resolved issues (50675)

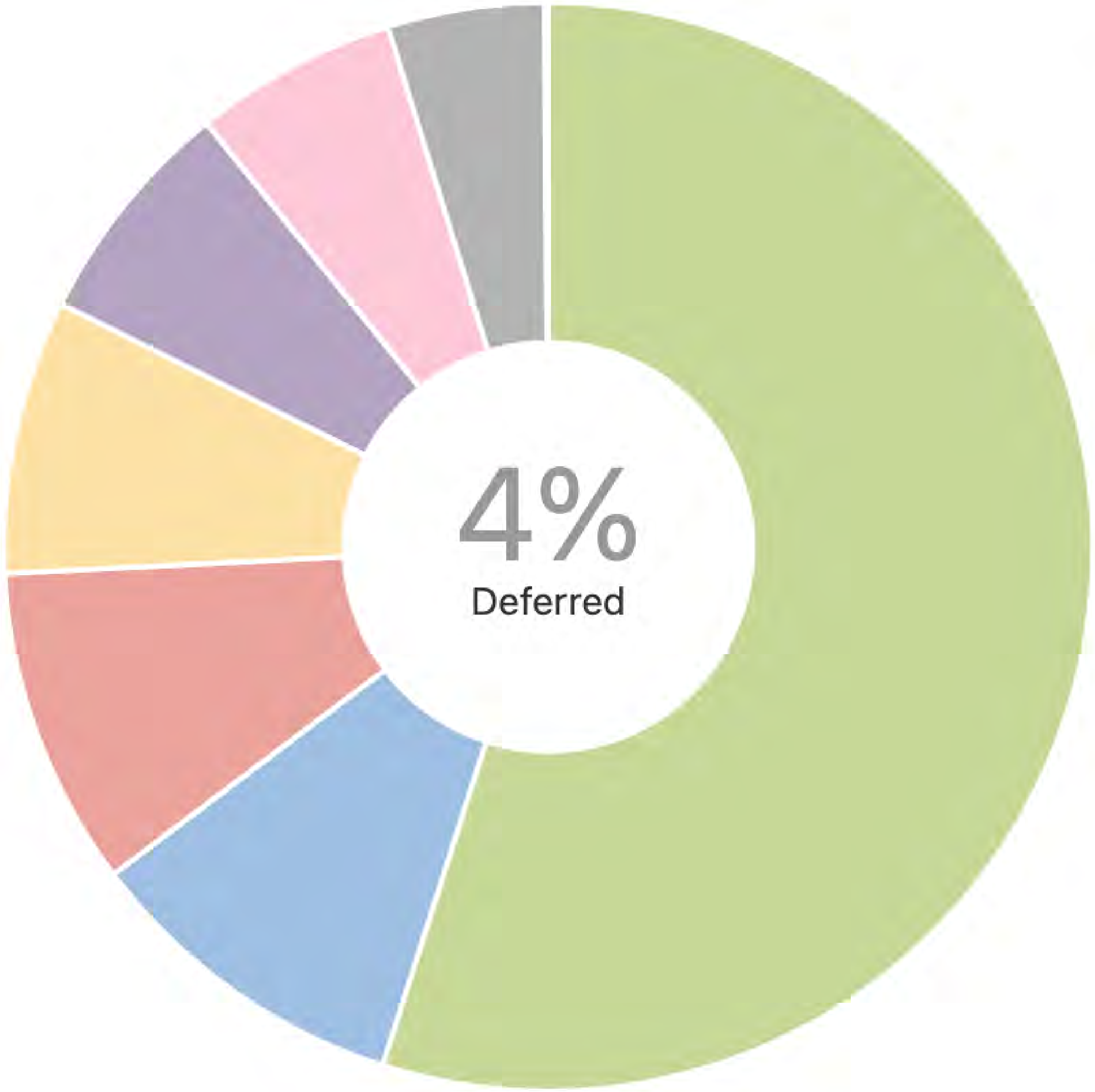

Resolution
Fixed
Unresolved
Won't Fix
Cannot Reproduce
Working As Designed
Duplicate
Deferred


54%
Fixed
Resolution
Fixed
Unresolved
Won't Fix
Cannot Reproduce
Working As Designed
Duplicate
Deferred



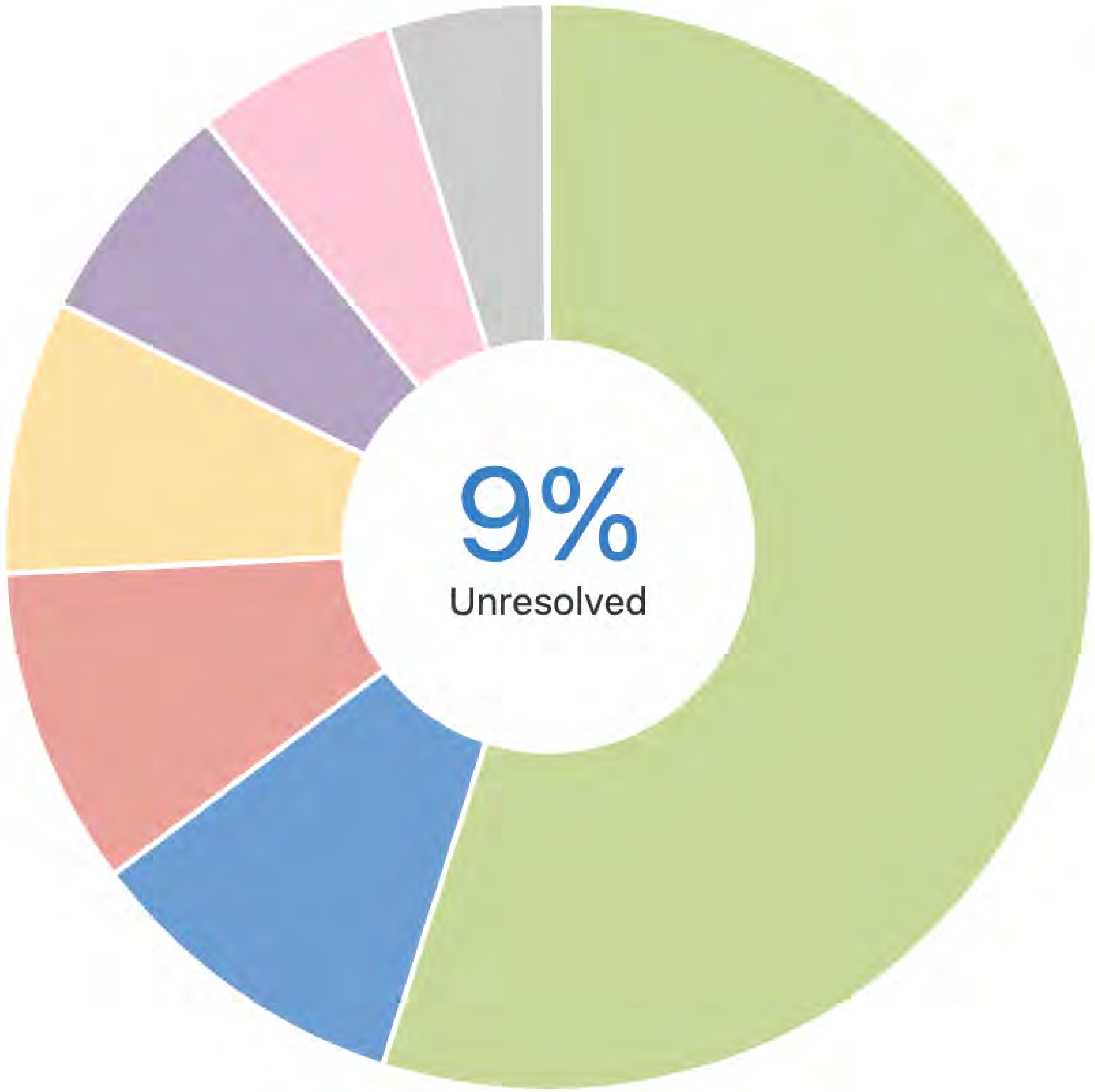
Resolution

🟩	Fixed
🟦	Unresolved
🟥	Won't Fix
🟨	Cannot Reproduce
🟪	Working As Designed
🟪	Duplicate
⬜	Deferred



Resolution

🟩	Fixed
🟦	Unresolved
🟥	Won't Fix
🟨	Cannot Reproduce
🟪	Working As Designed
🟥	Duplicate
⬜	Deferred


9%
Unresolved
Resolution
Fixed
Unresolved
Won't Fix
Cannot Reproduce
Working As Designed
Duplicate
Deferred

Answer: Iterative Refinement.

- Current programming methodologies lend themselves to iterative refinement
- We don’t solve problems; we approximate solutions

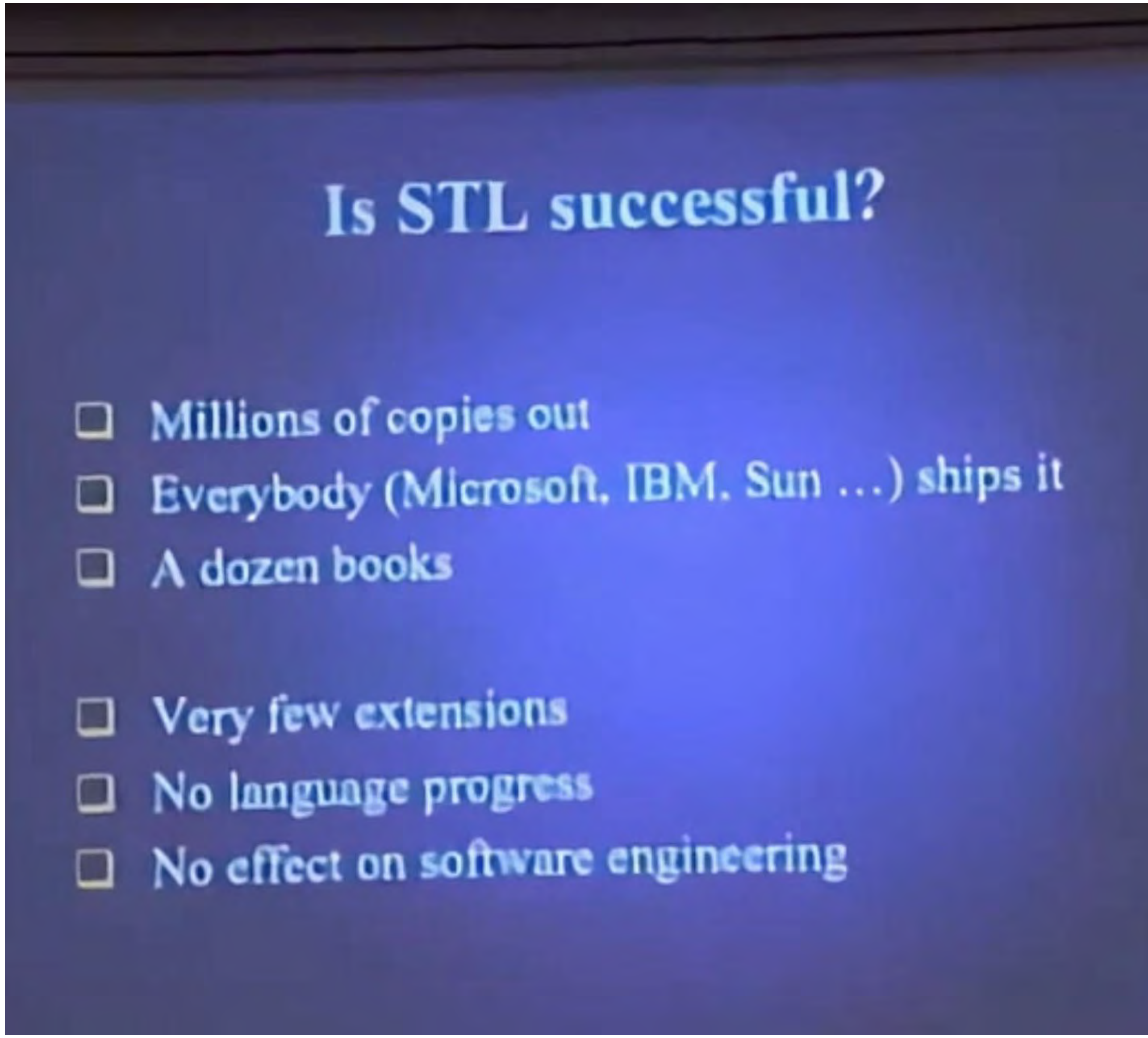
How can we write *Better Code*?

- Study how to write correct software
- Write algorithms once, in a general form that can be reused
- Focus on fundamental algorithms
- Compose larger systems from proven components

Generic Programming

Generic Programming

- In 2006, the idea of generic programming was still young
- In 2002, Alex Stepanov gave a presentation at Adobe with this slide



Is STL successful?

- ❑ Millions of copies out
- ❑ Everybody (Microsoft, IBM, Sun …) ships it
- ❑ A dozen books

- ❑ Very few extensions
- ❑ No language progress
- ❑ No effect on software engineering

“By generic programming we mean the definition of algorithms and data structures at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms.”

— David Musser & Alex Stepanov

"By **generic programming** we mean the definition of algorithms and data structures at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms."

— David Musser & Alex Stepanov

Adobe

“By generic programming we mean the definition of **algorithms** and **data structures** at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms.”

— David Musser & Alex Stepanov

Adobe

"By generic programming we mean the definition of algorithms and data structures at an **abstract** or **generic** level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms."

— David Musser & Alex Stepanov

“By generic programming we mean the definition of algorithms and data structures at an abstract or generic level, thereby **accomplishing** many **related programming tasks simultaneously**. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms.”

— David Musser & Alex Stepanov

"By generic programming we mean the definition of algorithms and data structures at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The **central notion** is that of **generic algorithms**, which are parameterized procedural schemata that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms."

— David Musser & Alex Stepanov

"By generic programming we mean the definition of algorithms and data structures at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are **parameterized procedural schemata** that are completely independent of the underlying data representation and are derived from concrete, efficient algorithms."

— David Musser & Alex Stepanov

Adobe

“By generic programming we mean the definition of algorithms and data structures at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely **independent** of the **underlying data representation** and are derived from concrete, efficient algorithms.”

— David Musser & Alex Stepanov

"By generic programming we mean the definition of algorithms and data structures at an abstract or generic level, thereby accomplishing many related programming tasks simultaneously. The central notion is that of generic algorithms, which are parameterized procedural schemata that are completely independent of the underlying data representation and are **derived from concrete, efficient algorithms**."

— David Musser & Alex Stepanov

Programming with Generics
≠
Generic Programming

Metaprogramming

≠

Generic Programming

Programming *is* Mathematics

- Generic Programming
    - Semantic Requirements
    - Concept
    - Model
    - Algorithms
    - Regular Function
    - Complexity

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

- Refined Concept: a concept defined by adding requirements to an existing concept
    - Monoid: a semigroup with an identity element
    - Bidirectional Iterator: forward iterator with constant complexity decrement
- Refined Algorithm: An algorithm performing the same function as another but with lower complexity or higher efficiency on a refined concept

Concepts

- A Concept is an algebraic structure formed of *connected* requirements
- Equality is a unique equivalence relation…

$$\forall a: a = a \text{ (reflexive)}$$
$$a = b \Longrightarrow b = a \text{ (symmetric)}$$
$$a = b \text{ and } b = c \Longrightarrow a = c \text{ (transitive)}$$

- …connected to copy and assignment:

$$b \rightarrow a \Longrightarrow a = b \text{ (copies are equal)}$$
$$a = b = c,\ d \neq a, d \rightarrow a \Longrightarrow b = c \text{ (copies are disjoint)}$$

- The concept *regular* is essential for equational and local reasoning

Building a Repository of Algorithms

- Modeled after the IRMT
- A collection of fundamental algorithms and data structures presented in a common form.
- Proven implementations in a variety of languages
- Implementations selected for inclusion based on abstractness and efficiency

Adobe

Unfortunately, the IRMT doesn't exist

- Building a cohesive foundation is *hard.*
- Examples:
  - *Euclid's Elements* (300 BCE)
  - *Principia Mathematica* by Isaac Newton (1687)
  - *Formulario Mathematico* by Giuseppe Peano (1894)
  - *Éléments de mathématique* by Nicolas Bourbaki (1939)

Computer Science foundations

- Example books:
  - *The Art of Computer Programming* by Don Knuth (1968..)
  - *Communicating Sequential Processes* by Tony Hoare (1978)
  - *Elements of Programming* by Alex Stepanov & Paul McJones (2019)

- Libraries:
  - Standard Template Library
  - Boost Graph Library
  - Ranges

The Language Challenge

- Semantic constraints are not expressed (and unchecked)
- Tradeoffs
  - Separate compilation makes refinement difficult (impacts efficiency and expressibility)
- Concept mechanisms are used for “implements” as opposed to “models”
  - See concepts like “std::copy_constructible”, or the trait “std::ops::Sub”, or the protocol “Hashable”
- Confusion over the domain of operations
  - A type *models* a concept if values of that type *exist* that satisfy the constraint
  - operator < on double models strict-weak-ordering

Adobe

Current Design of Large Systems

- Networks of objects form incidental data structures
- Messaging among objects forms incidental algorithms
- Design Patterns assist in reasoning about these systems
  - Local rules approximate correct algorithms and structures
- Iteratively refine until quality is *good enough*

A “Generic Algorithm” for Building Systems

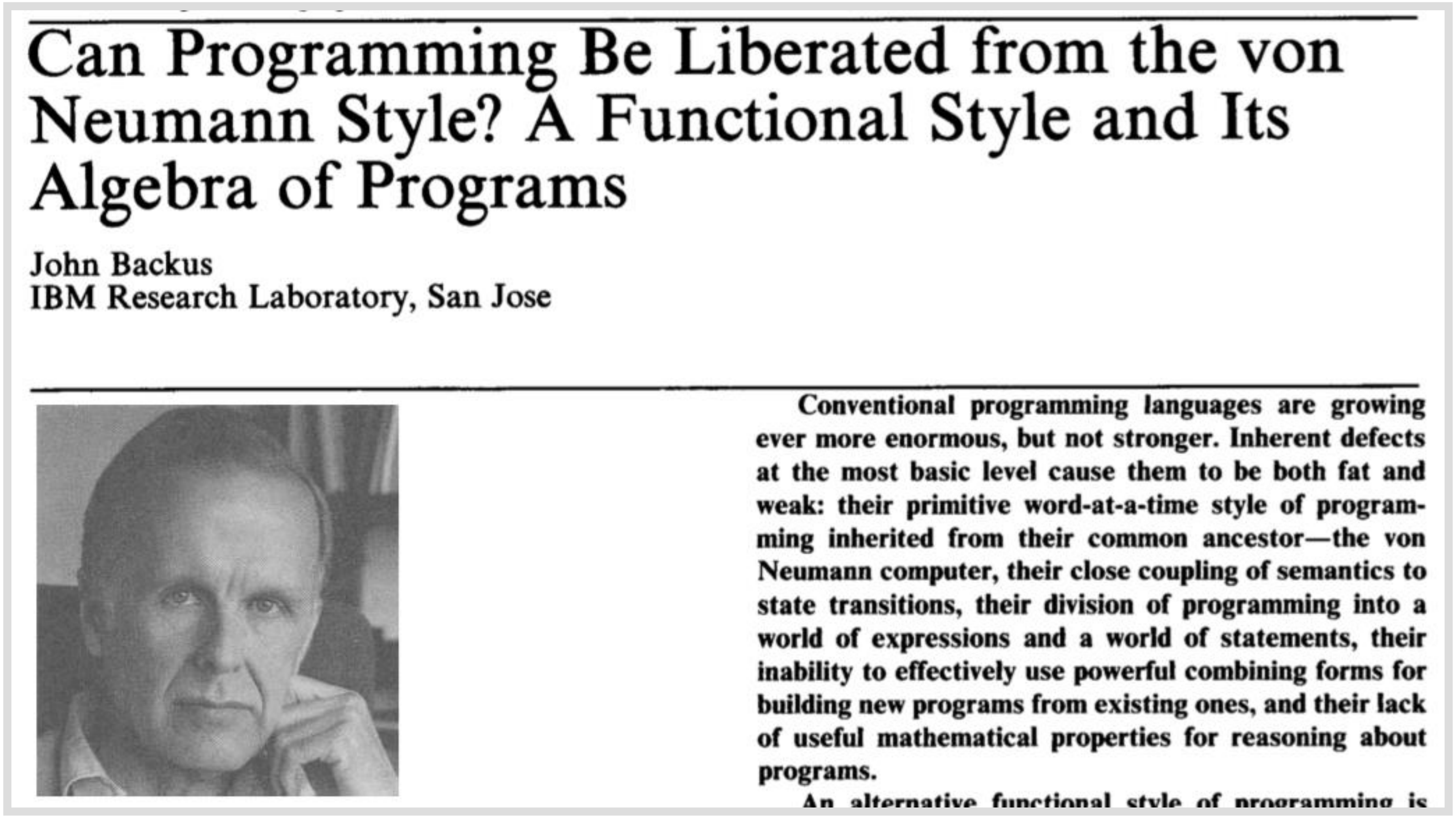
- Identify the components and how they connect and interrelate
- Avoid incidental data structures by packaging related objects, other than whole/part relationships, as parts of a whole
  - The whole maintains the invariants of any relationships between the parts
- Avoid incidental algorithms by recasting as explicit algorithms
- Identify common structures and algorithms, and refactor them into generic components
- Repeat…

Question: Is generic programming sufficient to build software at scale?

Conjecture: All problems of scale become a network problem

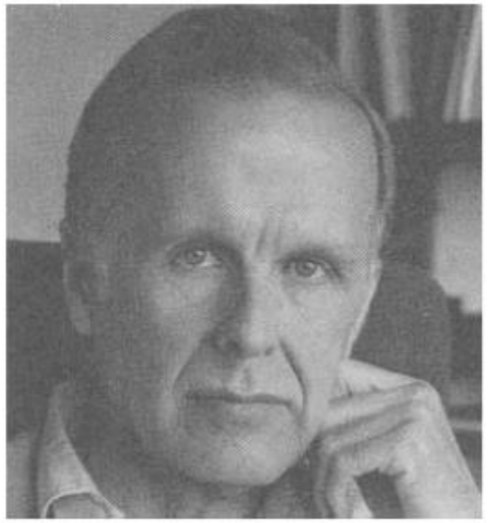
Declarative Forms

Declarative Forms



Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is

Declarative Forms

- Describe components by their structure and constraints in a domain-specific language
  - 4th generation 5th generation languages
- The DSLs should not strive to be general-purpose or Turing-complete
  - But Turing completeness, or effective Turing completeness, is nearly impossible to avoid
- Examples:
  - Lex and YACC (and BNF-based parser generators in general)
  - SQL
  - HTML (ignoring <script> tags)
  - Type Systems and Schema Languages
  - Spreadsheets

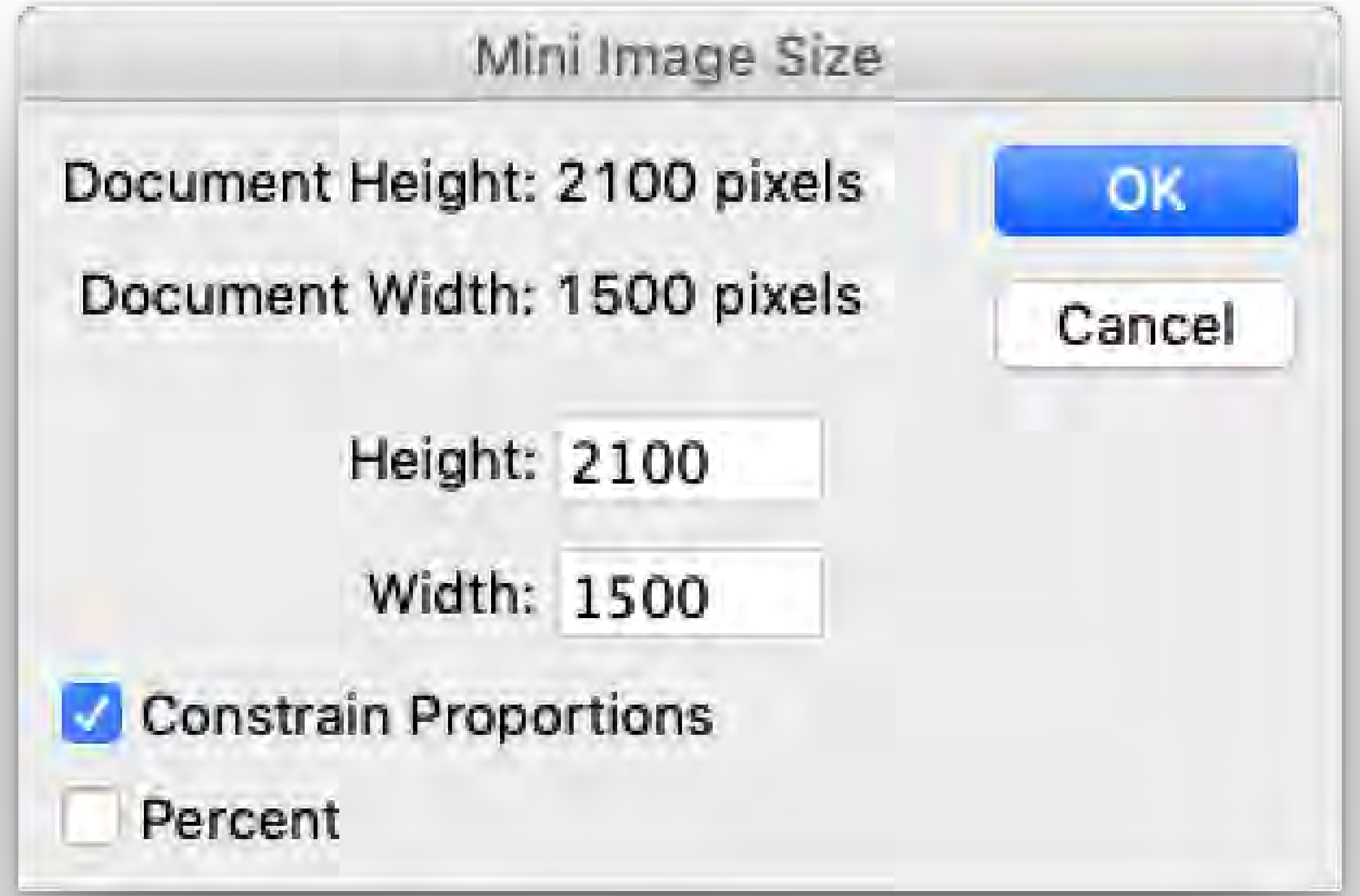
A “Generic Algorithm” for Building Large Systems

- Apply the algorithm for building systems
- Identify the common rules governing related parts of components and their structure
- Choose a DSL that allows you to define the component by expressing the structure and constraints
  - If a DSL doesn’t exist, develop one*
- Repeat

- *Currently *hard*

Rules Exist

Property Models


Mini Image Size

Document Height: 2100 pixels

Document Width: 1500 pixels

Height: 2100

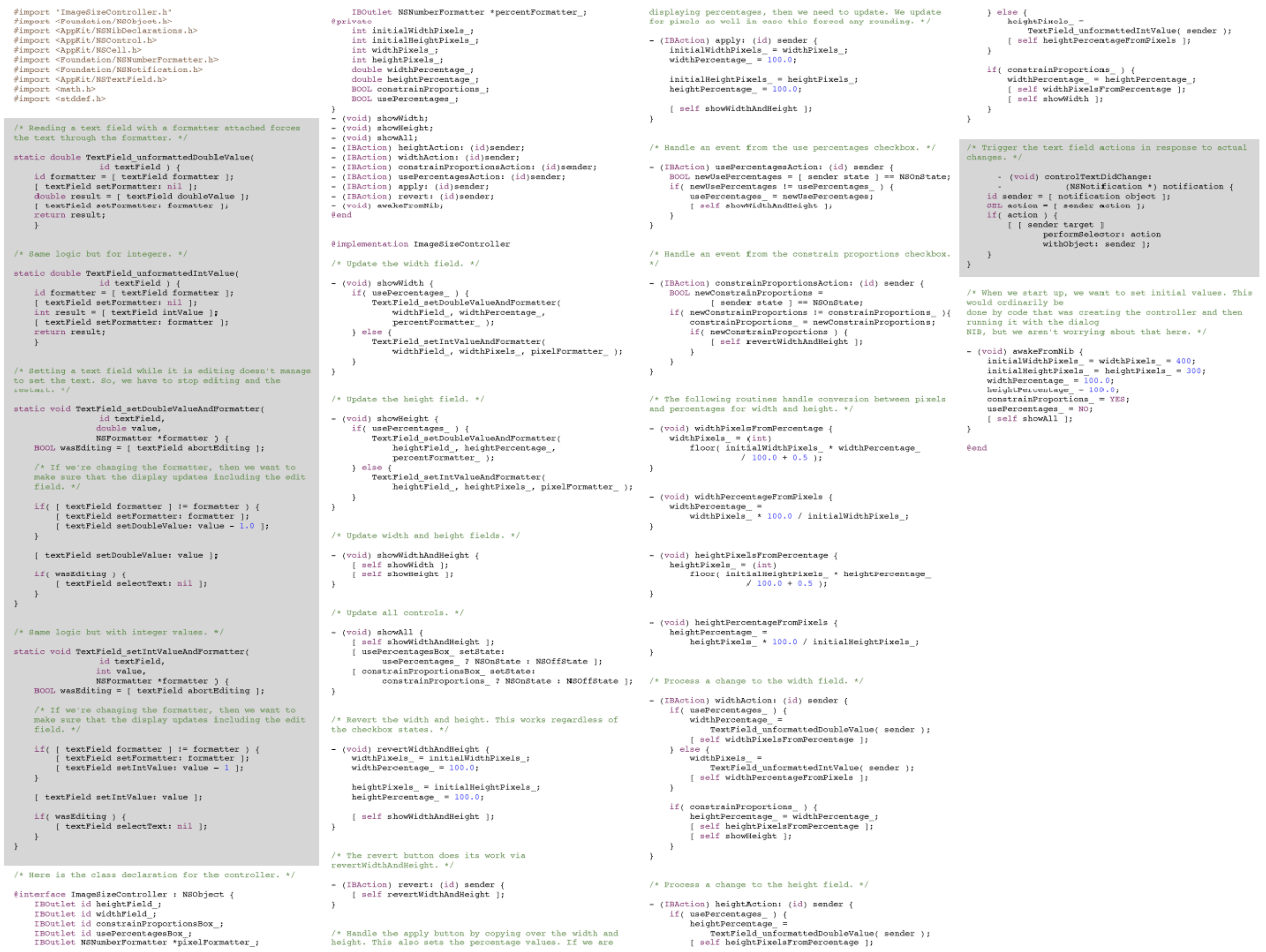
Width: 1500

[✓] Constrain Proportions

[ ] Percent

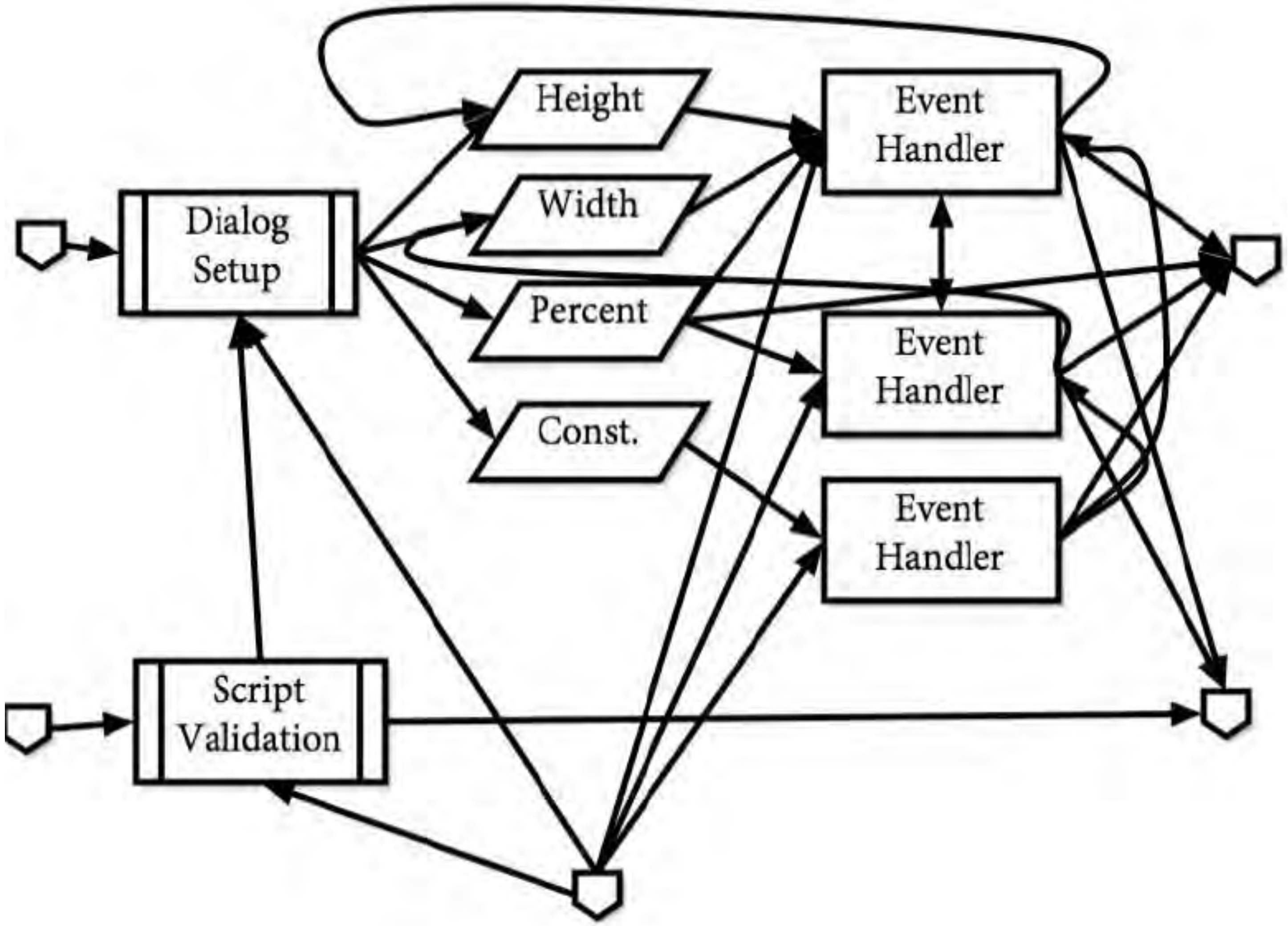
Adobe

Imperative Solution to Mini Image Size



```
#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
```

Event Flow


Height
Width
Percent
Const.
Dialog Setup
Script Validation
Event Handler
Event Handler
Event Handler

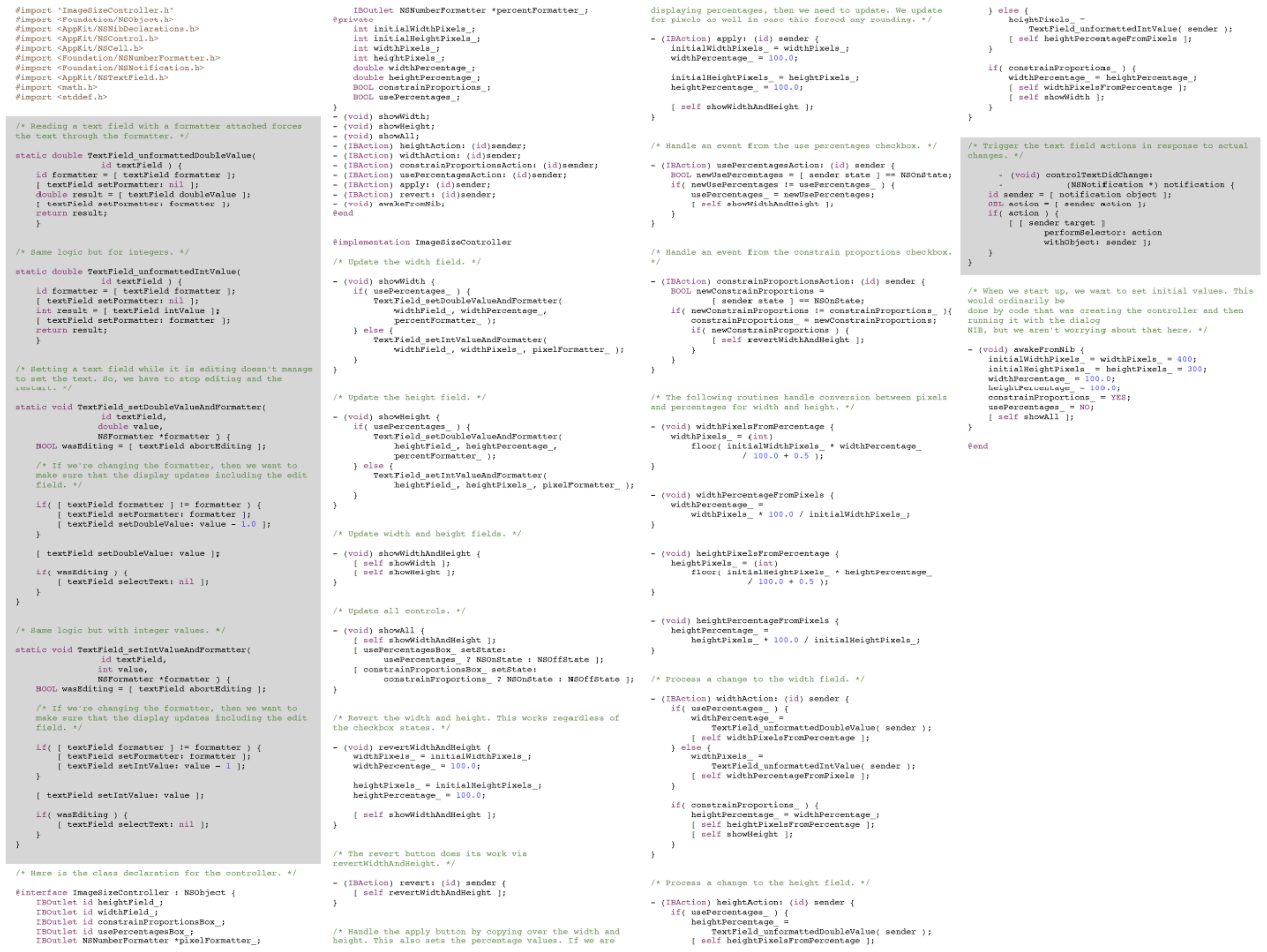
Demo

```
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 };
}
```

A Declarative Form

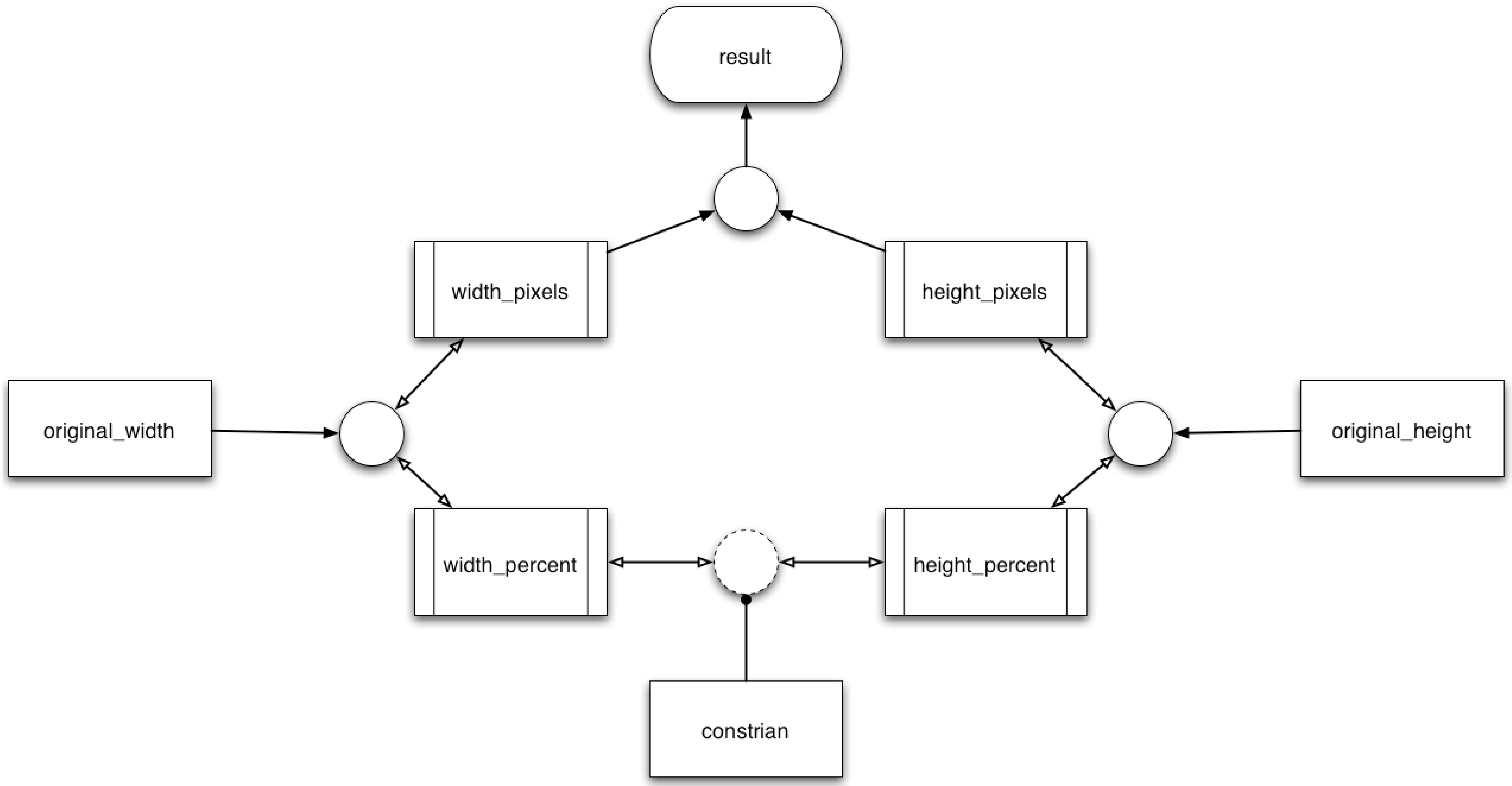
```
sheet mini_image_size(size: (usize, usize)) {
  var constrain_aspect_ratio: bool = true
  var pixel_size: (usize, usize) = size
  var percent: (f64, f64)

maintains:
  pixel_size.0 == round(percent.0 * size.0 / 100.0)
  pixel_size.1 == round(percent.1 * size.1 / 100.0)

  when constrain_aspect_ratio {
            percent.0 == percent.1
  }
}
```

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
```

Structure of Mini Image Size



Future of Software Development (2006 - restated)

- Extend the ideas of generic programming to more domains
- Extend generic programming to apply to runtime polymorphism
- Formally describe software behavior by expressing the structure and constraints of the system

References for Property Models

- Property models
  - *Generating Reactive Programs for Graphical User Interfaces from Multi-way Dataflow Constraint Systems* (2016), Foust G, Järvi J, Parent S
  - *Specializing Planners for Hierarchical Multi-way Dataflow Constraint Systems* (2015), Järvi J, Foust G, Haveraaen M
  - *HotDrink* A Library for Web User Interfaces (2013), Freeman J, Järvi J, Foust G
  - *Helping Programmers Help Users* (2012), Freeman J, Järvi J, Kim W, Marcus M, Parent S

Related Work

- Selections
  - One Way to Select Many (2016), Jaakko Järvi, Sean Parent
- Collection models
  - *Containers for GUI Models* (2024),
    Stokke, Knut Anders; Barash, Mikhail; Järvi, Jaakko; Stenholm, Elisabeth; Robbestad Gylterud, Håkon
  - *The Ultimate GUI Framework: Are We There Yet?* (2023), Stokke Knut Anders, Barash Mikhail, Järvi Jaakko
  - *A domain-specific language for structure manipulation in constraint system-based GUIs* (2023),
    Stokke Knut Anders, Barash Mikhail, Järvi Jaakko
  - *Towards Reusable GUI Structures* (2023), Stokke Knut Anders, Barash Mikhail, Järvi Jaakko

Future of Software Development

- Continue to improve the generic programming support in languages
  - Refinements
  - Dependent concepts
  - Law of exclusivity (follows from whole/part relationships and local reasoning)
  - Minimize tradeoffs between efficiency and safety
- Create better foundational libraries by implementing fundamental concepts and algorithms
- Create libraries of embedded DSLs that interoperate
- Develop AI to *reason* about code so that it can continue the above

Adobe