Avoiding Faulty User Interfaces

Jaakko Järvi

Feb 11th, 2014

Parasol
Smarter computing.
Texas A&M University

Programming Graphical User Interfaces (GUIs)

- UIs are perhaps the most costly area of software
- Observations in a major desktop software company:
  - 30+% of all code is in UI logic
  - 60+% of all defects in UI code

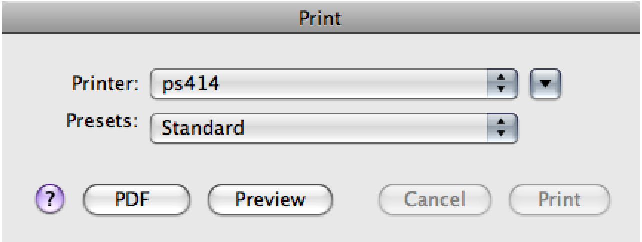
Programming Graphical User Interfaces (GUIs)

- UIs are perhaps the most costly area of software
- Observations in a major desktop software company:
  - 30+% of all code is in UI logic
  - 60+% of all defects in UI code
- GUI programming is not getting easier:
  - Multitude of platforms, devices, screen sizes, etc. to support
  - Responsiveness harder (latencies, failures in updating UI state)

Programming Graphical User Interfaces (GUIs)

- UIs are perhaps the most costly area of software
- Observations in a major desktop software company:
  - 30+% of all code is in UI logic
  - 60+% of all defects in UI code
- GUI programming is not getting easier:
  - Multitude of platforms, devices, screen sizes, etc. to support
  - Responsiveness harder (latencies, failures in updating UI state)
- The difficulty of UI programming underestimated

This is all too common

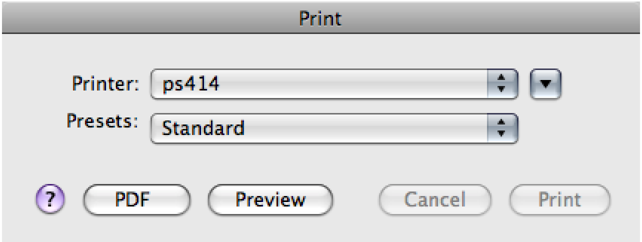


Print

Printer: ps414

Presets: Standard

This is all too common

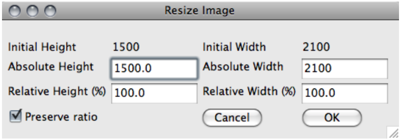


Printer: ps414

Presets: Standard

why?

GUI programming from the point of view of the developer



Resize Image

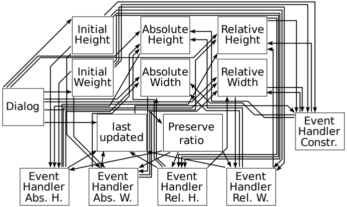
Initial Height 1500 Initial Width 2100

Absolute Height 1500.0 Absolute Width 2100

Relative Height (%) 100.0 Relative Width (%) 100.0

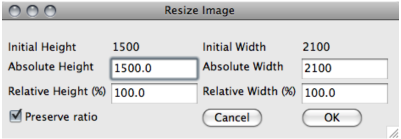
☑ Preserve ratio

Cancel OK



```
def ChangeCurrentHeightPx(self, event):
   self.LastUpdated = "Height"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current height, and compute relative height and place new rel. ht
   height = float(self.Controls["AbsolutePx"]["Height"].GetValue())
   pct = height / self.InitialSize[self.Height]
   self.Controls["Relative%"]["Height"].SetValue(str(pct*100))

   if constrained: # update width & width%
      self.Controls["Relative%"]["Width"].SetValue(str(pct*100))
      width = pct * self.InitialSize[self.Width]
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPx(self, event):
   self.LastUpdated = "Width"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current width, and compute relative width and place new rel. wd
   height = float(self.Controls["AbsolutePx"]["Width"].GetValue())
   pct = height / self.InitialSize[self.Width]
   self.Controls["Relative%"]["Width"].SetValue(str(pct*100))

   if constrained: # update height & height%
      self.Controls["Relative%"]["Height"].SetValue(str(pct*100))
      height = pct * self.InitialSize[self.Height]
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeCurrentHeightPct(self, event):
   self.LastUpdated = "Height"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current rel. ht, and compute absolute height and place new abs. ht
   height = float(self.Controls["Relative%"]["Height"].GetValue())
   cur = height * self.InitialSize[self.Height] / 100
   self.Controls["AbsolutePx"]["Height"].SetValue(str(round(cur)))

   if constrained: # update width & width%
      self.Controls["Relative%"]["Width"].SetValue(str(height))
      width = height * self.InitialSize[self.Width] / 100
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPct(self, event):
   self.LastUpdated = "Width"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current rel. wd, and compute absolute width and place new abs. wd
   width = float(self.Controls["Relative%"]["Width"].GetValue())
   cur = width * self.InitialSize[self.Width] / 100
   self.Controls["AbsolutePx"]["Width"].SetValue(str(round(cur)))

   if constrained: # update height & height%
      self.Controls["Relative%"]["Height"].SetValue(str(width))
      height = width * self.InitialSize[self.Height] / 100
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeConstrainState(self, event):
   constrained = self.Controls["Constrain"].GetValue()
   # If the ratio is constrained, determine which dimension
   # was last updated and update the OTHER dimension.
   # For example: if Height was last updated, use Height as
   # Width’s new percent, and update Width’s absolute value
   if constrained:
      if self.LastUpdated == "Height": # update width px & %
         pct = float(self.Controls["Relative%"]["Height"].GetValue())
         self.Controls["Relative%"]["Width"].SetValue(str(pct))
         width = pct * self.InitialSize[self.Width] / 100
         self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
      else: # update width px & %
         pct = float(self.Controls["Relative%"]["Width"].GetValue())
         self.Controls["Relative%"]["Height"].SetValue(str(pct))
         height = pct * self.InitialSize[self.Height] / 100
         self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
```

GUI programming from the point of view of the developer

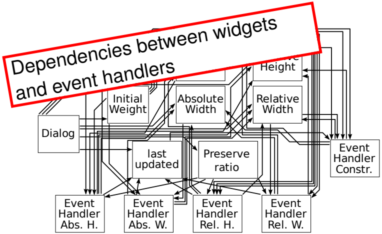


Initial Height 1500 Initial Width 2100

Absolute Height 1500.0 Absolute Width 2100

Relative Height (%) 100.0 Relative Width (%) 100.0

☑ Preserve ratio



Dependencies between widgets and event handlers

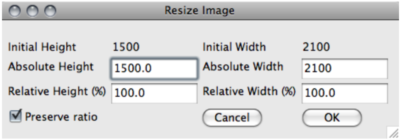
Typical event handling code (implementing most basic functionality)

```
def Cha
   self.La
   constra
   # no m                    percent & current stay bound together
   # get current height, and compute relative height and place new rel. ht
   height = float(self.Controls["AbsolutePx"]["Height"].GetValue())
   pct = height / self.InitialSize[self.Height]
   self.Controls["Relative%"]["Height"].SetValue(str(pct*100))

   if constrained: # update width & width%
      self.Controls["Relative%"]["Width"].SetValue(str(pct*100))
      width = pct * self.InitialSize[self.Width]
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPx(self, event):
   self.LastUpdated = "Width"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current width, and compute relative width and place new rel. wd
   height = float(self.Controls["AbsolutePx"]["Width"].GetValue())
   pct = height / self.InitialSize[self.Width]
   self.Controls["Relative%"]["Width"].SetValue(str(pct*100))

   if constrained: # update height & height%
      self.Controls["Relative%"]["Height"].SetValue(str(pct*100))
```

```
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeCurrentHeightPct(self, event):
   self.LastUpdated = "Height"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current rel. ht, and compute absolute height and place new abs. ht
   height = float(self.Controls["Relative%"]["Height"].GetValue())
   cur = height * self.InitialSize[self.Height] / 100
   self.Controls["AbsolutePx"]["Height"].SetValue(str(round(cur)))

   if constrained: # update width & width%
      self.Controls["Relative%"]["Width"].SetValue(str(height))
      width = height * self.InitialSize[self.Width] / 100
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPct(self, event):
   self.LastUpdated = "Width"
   constrained = self.Controls["Constrain"].GetValue()
   # no matter what the percent & current stay bound together
   # get current rel. wd, and compute absolute width and place new abs. wd
   width = float(self.Controls["Relative%"]["Width"].GetValue())
   cur = width * self.InitialSize[self.Width] / 100
   self.Controls["AbsolutePx"]["Width"].SetValue(str(round(cur)))
```

```
   if constrained: # update height & height%
      self.Controls["Relative%"]["Height"].SetValue(str(width))
      height = width * self.InitialSize[self.Height] / 100
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeConstrainState(self, event):
   constrained = self.Controls["Constrain"].GetValue()
   # If the ratio is constrained, determine which dimension
   # was last updated and update the OTHER dimension.
   # For example: if Height was last updated, use Height as
   # Width's new percent, and update Width's absolute value
   if constrained:
      if self.LastUpdated == "Height": # update width px & %
         pct = float(self.Controls["Relative%"]["Height"].GetValue())
         self.Controls["Relative%"]["Width"].SetValue(str(pct))
         width = pct * self.InitialSize[self.Width] / 100
         self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
      else: # update width px & %
         pct = float(self.Controls["Relative%"]["Width"].GetValue())
         self.Controls["Relative%"]["Height"].SetValue(str(pct))
         height = pct * self.InitialSize[self.Height] / 100
         self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
```

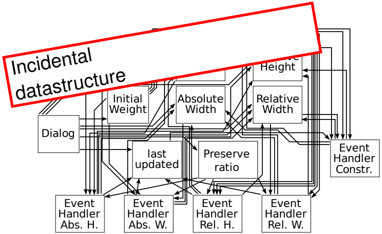
GUI programming from the point of view of the developer



Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0

☑ Preserve ratio



Incidental datastructure

Incidental algorithm

```
def Cha
  self.La
  constra
  # no m                  percent & current stay bound together
  # get current height, and compute relative height and place new rel. ht
  height = float(self.Controls["AbsolutePx"]["Height"].GetValue())
  pct = height / self.InitialSize[self.Height]
  self.Controls["Relative%"]["Height"].SetValue(str(pct*100))

  if constrained: # update width & width%
    self.Controls["Relative%"]["Width"].SetValue(str(pct*100))
    width = pct * self.InitialSize[self.Width]
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPx(self, event):
  self.LastUpdated = "Width"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current width, and compute relative width and place new rel. wd
  height = float(self.Controls["AbsolutePx"]["Width"].GetValue())
  pct = height / self.InitialSize[self.Width]
  self.Controls["Relative%"]["Width"].SetValue(str(pct*100))

  if constrained: # update height & height%
    self.Controls["Relative%"]["Height"].SetValue(str(pct*100))
```

```
                      Size[self.Height]
    self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeCurrentHeightPct(self, event):
  self.LastUpdated = "Height"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current rel. ht, and compute absolute height and place new abs. ht
  height = float(self.Controls["Relative%"]["Height"].GetValue())
  cur = height * self.InitialSize[self.Height] / 100
  self.Controls["AbsolutePx"]["Height"].SetValue(str(round(cur)))

  if constrained: # update width & width%
    self.Controls["Relative%"]["Width"].SetValue(str(height))
    width = height * self.InitialSize[self.Width] / 100
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPct(self, event):
  self.LastUpdated = "Width"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current rel. wd, and compute absolute width and place new abs. wd
  width = float(self.Controls["Relative%"]["Width"].GetValue())
  cur = width * self.InitialSize[self.Width] / 100
  self.Controls["AbsolutePx"]["Width"].SetValue(str(round(cur)))
```

```
  if constrained: # update height & height%
    self.Controls["Relative%"]["Height"].SetValue(str(width))
    height = width * self.InitialSize[self.Height] / 100
    self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeConstrainState(self, event):
  constrained = self.Controls["Constrain"].GetValue()
  # If the ratio is constrained, determine which dimension
  # was last updated and update the OTHER dimension.
  # For example: if Height was last updated, use Height as
  # Width's new percent, and update Width's absolute value
  if constrained:
    if self.LastUpdated == "Height": # update width px & %
      pct = float(self.Controls["Relative%"]["Height"].GetValue())
      self.Controls["Relative%"]["Width"].SetValue(str(pct))
      width = pct * self.InitialSize[self.Width] / 100
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
    else: # update width px & %
      pct = float(self.Controls["Relative%"]["Width"].GetValue())
      self.Controls["Relative%"]["Height"].SetValue(str(pct))
      height = pct * self.InitialSize[self.Height] / 100
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
```

Observation 1

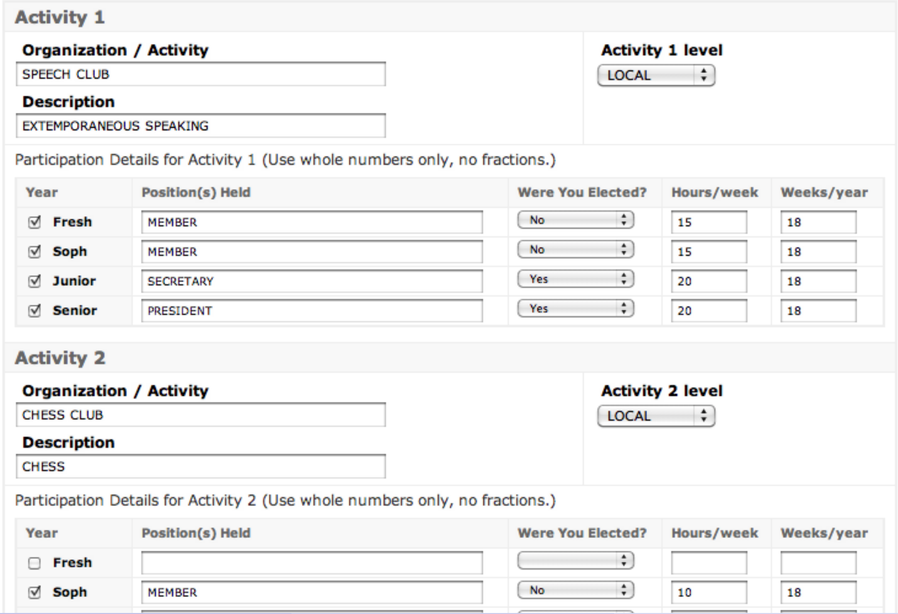
Programming user interfaces constitutes a significant portion of all programming effort.

GUI programming from the point of view of the user

GUI programming from the point of view of the user

POOR QUALITY GUIS ARE FRUSTRATING!

A concrete instance of GUI frustration



Activity 1

Organization / Activity

SPEECH CLUB

Description

EXTEMPORANEOUS SPEAKING

Activity 1 level

LOCAL

Participation Details for Activity 1 (Use whole numbers only, no fractions.)

Year	Position(s) Held	Were You Elected?	Hours/week	Weeks/year
☑ **Fresh**	MEMBER	No	15	18
☑ **Soph**	MEMBER	No	15	18
☑ **Junior**	SECRETARY	Yes	20	18
☑ **Senior**	PRESIDENT	Yes	20	18

Activity 2

Organization / Activity

CHESS CLUB

Description

CHESS

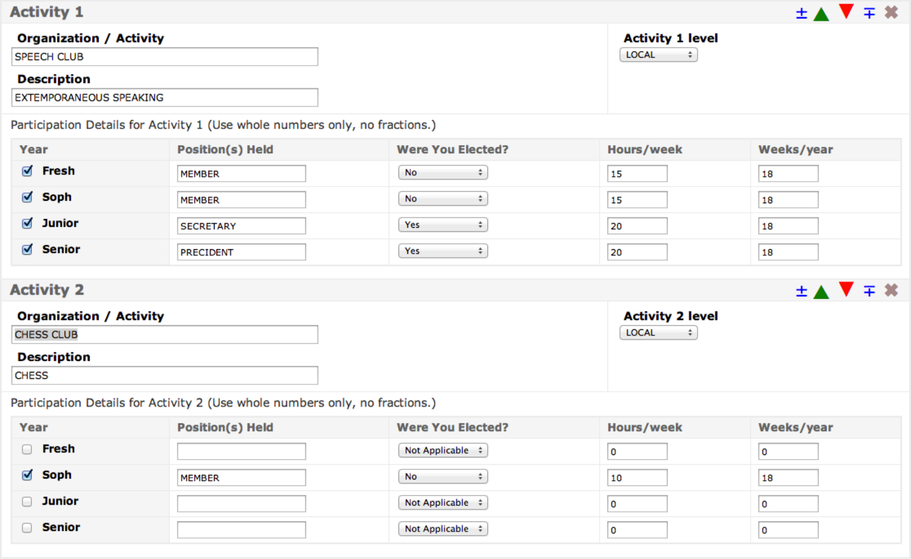
Activity 2 level

LOCAL

Participation Details for Activity 2 (Use whole numbers only, no fractions.)

Year	Position(s) Held	Were You Elected?	Hours/week	Weeks/year
□ **Fresh**				
☑ **Soph**	MEMBER	No	10	18

A concrete instance of GUI frustration



Activity 1

Organization / Activity

SPEECH CLUB

Description

EXTEMPORANEOUS SPEAKING

Activity 1 level

LOCAL

Participation Details for Activity 1 (Use whole numbers only, no fractions.)

Year	Position(s) Held	Were You Elected?	Hours/week	Weeks/year
☑ Fresh	MEMBER	No	15	18
☑ Soph	MEMBER	No	15	18
☑ Junior	SECRETARY	Yes	20	18
☑ Senior	PRECIDENT	Yes	20	18

Activity 2

Organization / Activity

CHESS CLUB

Description

CHESS

Activity 2 level

LOCAL

Participation Details for Activity 2 (Use whole numbers only, no fractions.)

Year	Position(s) Held	Were You Elected?	Hours/week	Weeks/year
☐ Fresh		Not Applicable	0	0
☑ Soph	MEMBER	No	10	18
☐ Junior		Not Applicable	0	0
☐ Senior		Not Applicable	0	0

Back of the envelope estimate

- Fact: Roughly 250,000 high-school graduates each year
- Guess: 125,000 uses of `www.applytexas.org`
- Guess: 60,000 need to re-order extracurricular activities
- Time invested:

$$60,000 \times 5 \text{ mins} = 300,000 \text{ mins}$$

Back of the envelope estimate

- Fact: Roughly 250,000 high-school graduates each year
- Guess: 125,000 uses of `www.applytexas.org`
- Guess: 60,000 need to re-order extracurricular activities
- Time invested:

$$\begin{aligned} 60,000 \times 5 \text{ mins} &= 300,000 \text{ mins} \\ &\sim 208 \text{ days} \end{aligned}$$

Back of the envelope estimate

- Fact: Roughly 250,000 high-school graduates each year
- Guess: 125,000 uses of `www.applytexas.org`
- Guess: 60,000 need to re-order extracurricular activities
- Time invested:

$$\begin{gathered} 60,000 \times 5 \text{ mins} = 300,000 \text{ mins} \\ \sim 208 \text{ days} \\ > \tfrac{1}{2} \text{ year} \end{gathered}$$

Back of the envelope estimate

- Fact: Roughly 250,000 high-school graduates each year
- Guess: 125,000 uses of `www.applytexas.org`
- Guess: 60,000 need to re-order extracurricular activities
- Time invested:

$$\begin{gathered}60,000 \times 5 \text{ mins} = 300,000 \text{ mins} \\ \sim 208 \text{ days} \\ > \tfrac{1}{2} \text{ year} \\ \sim 2 \text{ developer years}\end{gathered}$$

Impact of a nuisance

- ApplyTexas.org is just one little app in one corner of the world, but the same repeats everywhere
  - e-commerce sites
  - travel bookings
  - tax form preparation software
  - “in-house” business applications
  - even high-end desktop applications

Impact of a nuisance

- ApplyTexas.org is just one little app in one corner of the world, but the same repeats everywhere
  - e-commerce sites
  - travel bookings
  - tax form preparation software
  - “in-house” business applications
  - even high-end desktop applications

- A small waste of effort significant when aggregated over a large number of users

Impact of a nuisance

- ApplyTexas.org is just one little app in one corner of the world, but the same repeats everywhere
  - e-commerce sites
  - travel bookings
  - tax form preparation software
  - “in-house” business applications
  - even high-end desktop applications

- A small waste of effort significant when aggregated over a large number of users

- A small waste of effort significant <u>even for one user</u> when repeated in many user interfaces or by repeated use of one

Observation 2

Poor quality of user interfaces contribute to a significant waste of human effort

Why everything is broken and nobody's upset

- Users experience low quality in small doses, too small to complain
- An individual user's reaction to a usability problem
  - grumbling
  - attempt to find a work-around
  - succeed or give up
  - soldier on
- Perceived per user cost of low quality is low
- Per developer cost of eliminating frustration is high(er)

Why everything is broken and nobody's upset

- Users experience low quality in small doses, too small to complain
- An individual user's reaction to a usability problem
  - grumbling
  - attempt to find a work-around
  - succeed or give up
  - soldier on
- Perceived per user cost of low quality is low
- Per developer cost of eliminating frustration is high(er)
- This imbalance rewards producing barely passable quality

Why everything is broken and nobody's upset

- Users experience low quality in small doses, too small to complain
- An individual user's reaction to a usability problem
  - grumbling
  - attempt to find a work-around
  - succeed or give up
  - soldier on
- Perceived per user cost of low quality is low
- Per developer cost of eliminating frustration is high(er)
- This imbalance rewards producing barely passable quality
- Even if this was not the case, programming feature-rich and correct UIs is not easy at all (▸ demo)

Simple UI? I

Some considerations for the UI programmer:

- Which fields need to be recomputed and to which values after a change
- Should some widgets be disabled or enabled after an interaction
- Indicate that a value is pending if there is a delay
- Keep the UI responsive even though some values are pending
- Keep updates consistent and cancel unnecessary computations in case interactions happen while computation is ongoing
- Invalid inputs should be rejected or indicated somehow
- Helpful error messages should be given to the user, pointing accurately where troublesome values are
- Failed computations by the user interface should be handled, and the reasons communicated through helpful error messages
- Undo/redo

Simple UI? II

- Copy/paste
- Reacting to external changes (change of window size, abruptly closing the window)
- Support both mouse and keyboard navigation
- The UI should support *scripting*

Algorithms for User Interfaces

Ideal

- Developing a high-quality feature-rich GUI is no more expensive than developing a low-quality bare-bones GUI.

Algorithms for User Interfaces

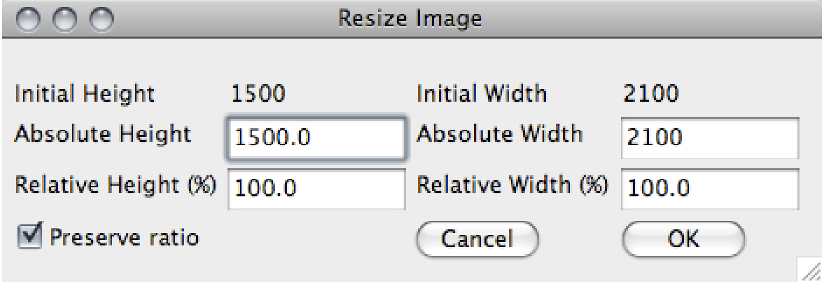
Ideal

- Developing a high-quality feature-rich GUI is no more expensive than developing a low-quality bare-bones GUI.

Approach

- Declarative programming, constraint systems
  - Specify dependencies amongst data in a GUI as a *hierarchical multi-way data-flow constraint system*
  - A non-incidental real data structure
- GUI behaviors are reusable algorithms over the constraint system data structure
  - updating values, enabling/disabling widgets, scripting, undo/redo, spinners for pending values, responsiveness, pinning values, accurate error messages, ...

Model for UIs: Data with constraints



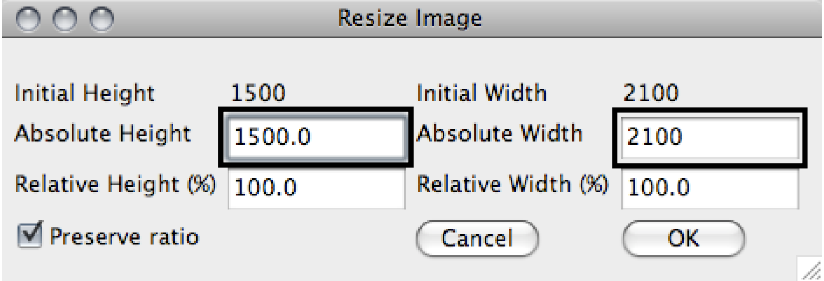
Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0

☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

Model for UIs: Data with constraints

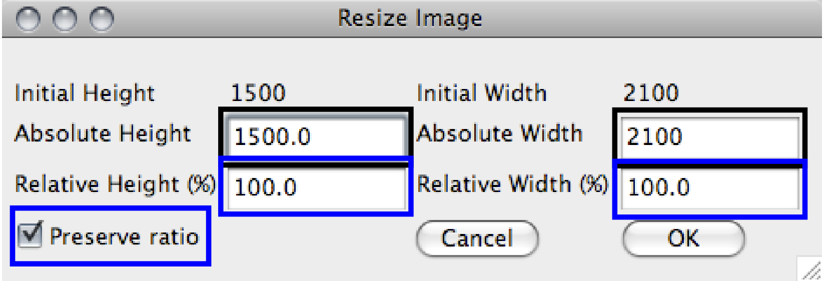


Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0

☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

Model for UIs: Data with constraints



Resize Image

Initial Height 1500

Initial Width 2100

Absolute Height 1500.0

Absolute Width 2100

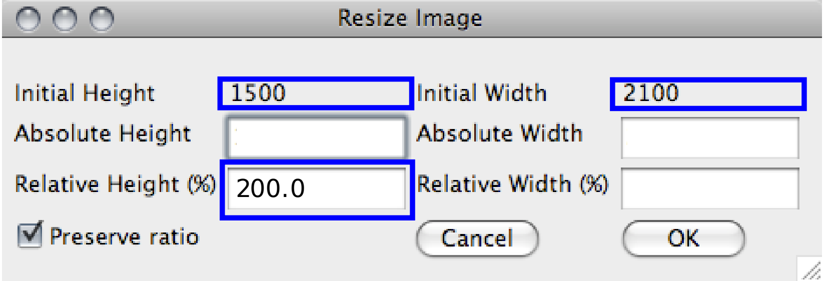
Relative Height (%) 100.0

Relative Width (%) 100.0

☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

Model for UIs: Data with constraints



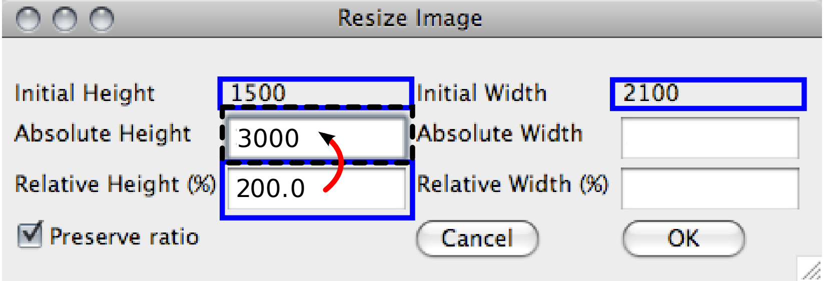
Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height		Absolute Width	
Relative Height (%)	200.0	Relative Width (%)	

☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

Model for UIs: Data with constraints



Resize Image

Initial Height 1500

Initial Width 2100

Absolute Height 3000

Absolute Width

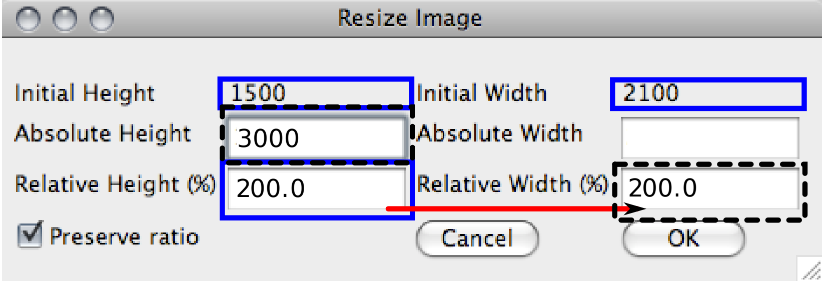
Relative Height (%) 200.0

Relative Width (%)

☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

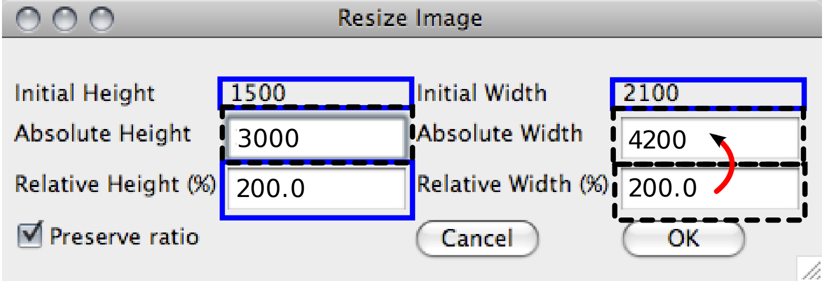
Model for UIs: Data with constraints



Initial Height 1500
Initial Width 2100
Absolute Height 3000
Absolute Width
Relative Height (%) 200.0
Relative Width (%) 200.0
☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

Model for UIs: Data with constraints



Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height	3000	Absolute Width	4200
Relative Height (%)	200.0	Relative Width (%)	200.0

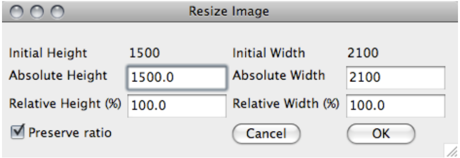
☑ Preserve ratio

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency

Model for UIs: Multi-way dataflow constraint system

View

Model



Resize Image

Initial Height 1500

Initial Width 2100

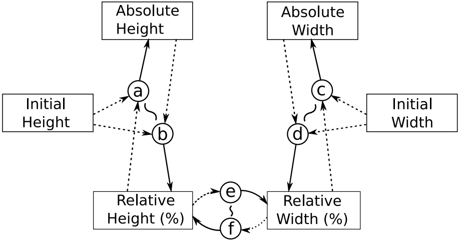
Absolute Height 1500.0

Absolute Width 2100

Relative Height (%) 100.0

Relative Width (%) 100.0

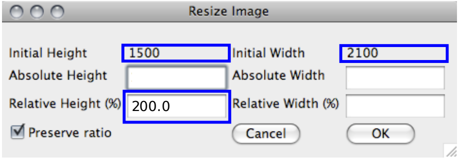
☑ Preserve ratio



Model for UIs: Multi-way dataflow constraint system

View

Model



Resize Image

Initial Height 1500

Initial Width 2100

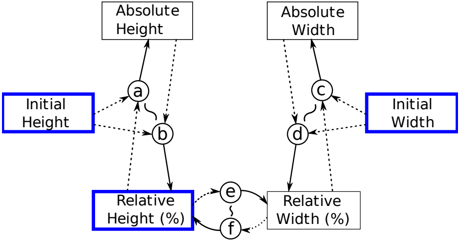
Absolute Height

Absolute Width

Relative Height (%) 200.0

Relative Width (%)

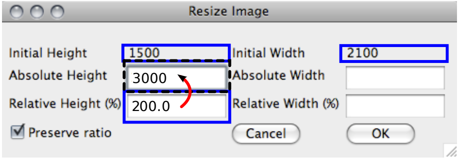
☑ Preserve ratio



Model for UIs: Multi-way dataflow constraint system

View

Model



Resize Image

Initial Height 1500

Initial Width 2100

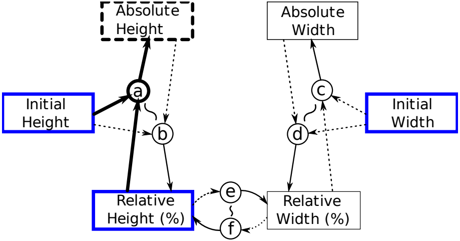
Absolute Height 3000

Absolute Width

Relative Height (%) 200.0

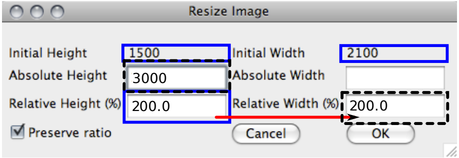
Relative Width (%)

☑ Preserve ratio



Model for UIs: Multi-way dataflow constraint system

View

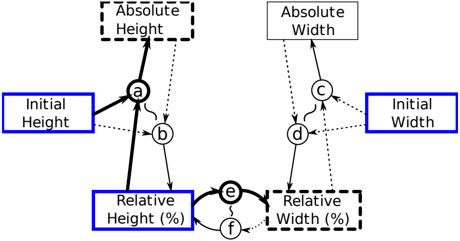


Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height	3000	Absolute Width	
Relative Height (%)	200.0	Relative Width (%)	200.0

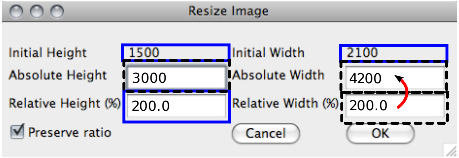
☑ Preserve ratio

Model



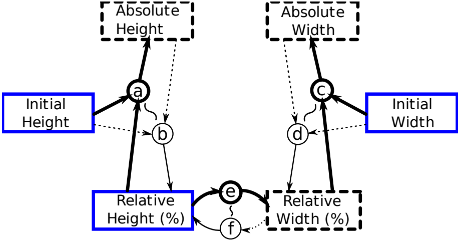
Model for UIs: Multi-way dataflow constraint system

View


Resize Image

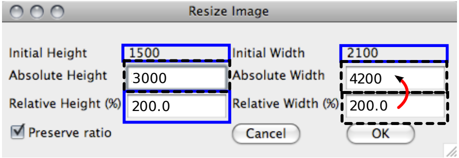
Initial Height 1500
Initial Width 2100
Absolute Height 3000
Absolute Width 4200
Relative Height (%) 200.0
Relative Width (%) 200.0
☑ Preserve ratio

Model



Model for UIs: Multi-way dataflow constraint system

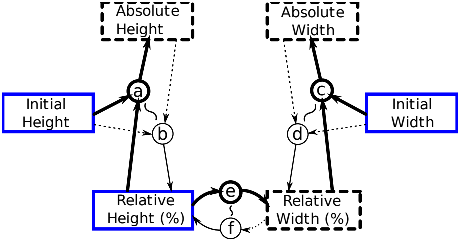
View



Resize Image

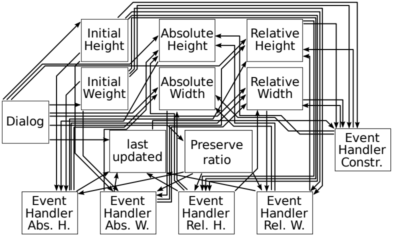
Initial Height 1500
Initial Width 2100
Absolute Height 3000
Absolute Width 4200
Relative Height (%) 200.0
Relative Width (%) 200.0
☑ Preserve ratio

Model

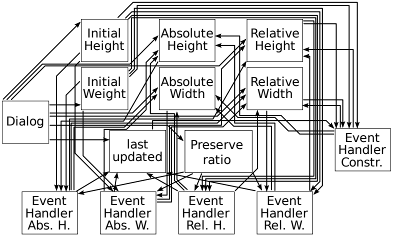


- Event handler code for “onChange” event in a view trivial:
  1. update a variable in the constraint system
  2. solve
  3. other views update their values

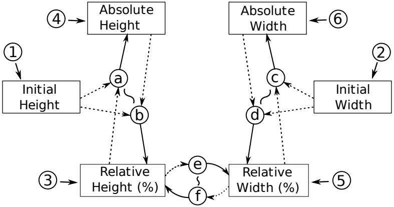
Incidental Data Structure → Explicit Model



Incidental Data Structure → Explicit Model



⟹



Code of Incidental Algorithm → Model Declaration

```
def ChangeCurrentHeightPx(self, event):
  self.LastUpdated = "Height"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current height, and compute relative height and place new rel. ht
  height = float(self.Controls["AbsolutePx"]["Height"].GetValue())
  pct = height / self.InitialSize[self.Height]
  self.Controls["Relative%"]["Height"].SetValue(str(pct*100))

  if constrained: # update width & width%
    self.Controls["Relative%"]["Width"].SetValue(str(pct*100))
    width = pct * self.InitialSize[self.Width]
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPx(self, event):
  self.LastUpdated = "Width"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current width, and compute relative width and place new rel. wd
  height = float(self.Controls["AbsolutePx"]["Width"].GetValue())
  pct = height / self.InitialSize[self.Width]
  self.Controls["Relative%"]["Width"].SetValue(str(pct*100))

  if constrained: # update height & height%
    self.Controls["Relative%"]["Height"].SetValue(str(pct*100))
    height = pct * self.InitialSize[self.Height]
    self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeCurrentHeightPct(self, event):
  self.LastUpdated = "Height"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current rel. ht, and compute absolute height and place new abs. ht
  height = float(self.Controls["Relative%"]["Height"].GetValue())
  cur = height * self.InitialSize[self.Height] / 100
  self.Controls["AbsolutePx"]["Height"].SetValue(str(round(cur)))

  if constrained: # update width & width%
    self.Controls["Relative%"]["Width"].SetValue(str(height))
    width = height * self.InitialSize[self.Width] / 100
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPct(self, event):
  self.LastUpdated = "Width"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current rel. wd, and compute absolute width and place new abs. wd
  width = float(self.Controls["Relative%"]["Width"].GetValue())
  cur = width * self.InitialSize[self.Width] / 100
  self.Controls["AbsolutePx"]["Width"].SetValue(str(round(cur)))

  if constrained: # update height & height%
    self.Controls["Relative%"]["Height"].SetValue(str(width))
    height = width * self.InitialSize[self.Height] / 100
    self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeConstrainState(self, event):
  constrained = self.Controls["Constrain"].GetValue()
  # If the ratio is constrained, determine which dimension
  # was last updated and update the OTHER dimension.
  # For example: if Height was last updated, use Height as
  # Width's new percent, and update Width's absolute value
  if constrained:
    if self.LastUpdated == "Height": # update width px & %
      pct = float(self.Controls["Relative%"]["Height"].GetValue())
      self.Controls["Relative%"]["Width"].SetValue(str(pct))
      width = pct * self.InitialSize[self.Width] / 100
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
    else: # update width px & %
      pct = float(self.Controls["Relative%"]["Width"].GetValue())
      self.Controls["Relative%"]["Height"].SetValue(str(pct))
      height = pct * self.InitialSize[self.Height] / 100
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
```

Code of Incidental Algorithm → Model Declaration

```
def ChangeCurrentHeightPx(self, event):
  self.LastUpdated = "Height"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current height, and compute relative height and place new rel. ht
  height = float(self.Controls["AbsolutePx"]["Height"].GetValue())
  pct = height / self.InitialSize[self.Height]
  self.Controls["Relative%"]["Height"].SetValue(str(pct∗100))

  if constrained: # update width & width%
    self.Controls["Relative%"]["Width"].SetValue(str(pct∗100))
    width = pct ∗ self.InitialSize[self.Width]
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPx(self, event):
  self.LastUpdated = "Width"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current width, and compute relative width and place new rel. wd
  height = float(self.Controls["AbsolutePx"]["Width"].GetValue())
  pct = height / self.InitialSize[self.Width]
  self.Controls["Relative%"]["Width"].SetValue(str(pct∗100))

  if constrained: # update height & height%
    self.Controls["Relative%"]["Height"].SetValue(str(pct∗100))
    height = pct ∗ self.InitialSize[self.Height]
    self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeCurrentHeightPct(self, event):
  self.LastUpdated = "Height"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current rel. ht, and compute absolute height and place new abs. ht
  height = float(self.Controls["Relative%"]["Height"].GetValue())
  cur = height ∗ self.InitialSize[self.Height] / 100
  self.Controls["AbsolutePx"]["Height"].SetValue(str(round(cur)))

  if constrained: # update width & width%
    self.Controls["Relative%"]["Width"].SetValue(str(height))
    width = height ∗ self.InitialSize[self.Width] / 100
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentWidthPct(self, event):
  self.LastUpdated = "Width"
  constrained = self.Controls["Constrain"].GetValue()
  # no matter what the percent & current stay bound together
  # get current rel. wd, and compute absolute width and place new abs. wd
  width = float(self.Controls["Relative%"]["Width"].GetValue())
  cur = width ∗ self.InitialSize[self.Width] / 100
  self.Controls["AbsolutePx"]["Width"].SetValue(str(round(cur)))

  if constrained: # update height & height%
    self.Controls["Relative%"]["Height"].SetValue(str(width))
    height = width ∗ self.InitialSize[self.Height] / 100
    self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

def ChangeConstrainState(self, event):
  constrained = self.Controls["Constrain"].GetValue()
  # If the ratio is constrained, determine which dimension
  # was last updated and update the OTHER dimension.
  # For example: if Height was last updated, use Height as
  # Width's new percent, and update Width's absolute value
  if constrained:
    if self.LastUpdated == "Height": # update width px & %
      pct = float(self.Controls["Relative%"]["Height"].GetValue())
      self.Controls["Relative%"]["Width"].SetValue(str(pct))
      width = pct ∗ self.InitialSize[self.Width] / 100
      self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
    else: # update width px & %
      pct = float(self.Controls["Relative%"]["Width"].GetValue())
      self.Controls["Relative%"]["Height"].SetValue(str(pct))
      height = pct ∗ self.InitialSize[self.Height] / 100
      self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
```

⇒

```
sheet image_resize {
  input:
    initial_width  : 5 ∗ 300;
    initial_height : 7 ∗ 300;
  interface:
    preserve_ratio : true;
    absolute_width  : initial_width;
    absolute_height : initial_height;
    relative_width; relative_height;
  logic:
    relate {
      absolute_height <== relative_height ∗ initial_height / 100;
      relative_height <== absolute_height ∗ 100 / initial_height;
    }
    relate {
      absolute_width <== relative_width ∗ initial_width / 100;
      relative_width <== absolute_width ∗ 100 / initial_width;
    }
    when (preserve_ratio) relate {
      relative_width <== relative_height;
      relative_height <== relative_width;
    }
}
```

```
sheet image_resize {
  input:
    initial_width  : 5 * 300;
    initial_height : 7 * 300;
  interface:
    preserve_ratio : true;
    absolute_width  : initial_width;
    absolute_height : initial_height;
    relative_width; relative_height;
  logic:
    relate {
      absolute_height <== relative_height * initial_height / 100;
      relative_height <== absolute_height * 100 / initial_height;
    }
    relate {
      absolute_width <== relative_width * initial_width / 100;
      relative_width <== absolute_width * 100 / initial_width;
    }
    when (preserve_ratio) relate {
      relative_width <== relative_height;
      relative_height <== relative_width;
    }
}
```

Key obsrevation

Reifying the dependencies enables reusable GUI algorithms.

Key obsrevation

Reifying the dependencies enables reusable GUI algorithms.

- Examples:
    - can a variable impact an output?
    - is a variable pending?

Key obsrevation

Reifying the dependencies enables reusable GUI algorithms.

- Examples:
  - can a variable impact an output?
  - is a variable pending?

▸ Stocks demo

▸ Stocks source

Experiences adopting property models

- UI behaviors included
  - Maintaining consistency (updating widget values)
  - Widget enablement/disablement
  - Command activation/deactivation
  - Scripting
- Code reduction of 8—10 to one in statement counts
- Improved quality
  - Fewer defects
  - Consistency among different user interface
  - More features
- Anecdote: impact on a single dialog's event handling and scripting code
  - Before: 781 statements, 5 known logic defects
  - After: 46 statements, no known defects

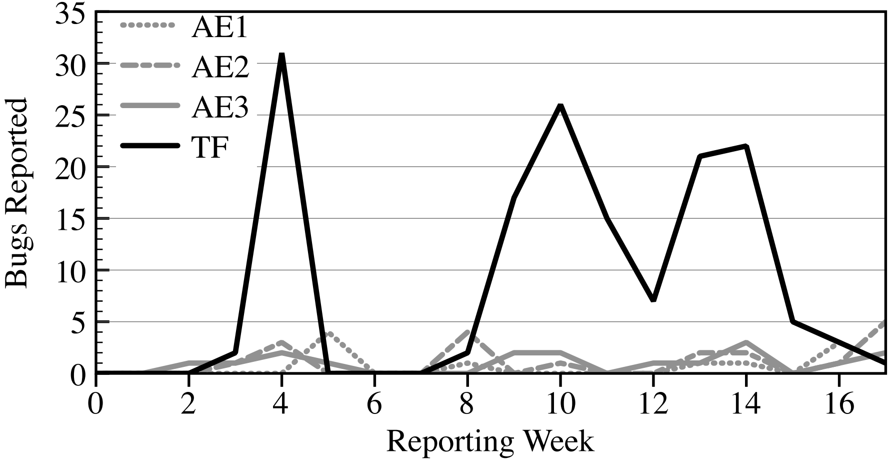
Experiment

- Rewriting user interface code for a major desktop application
- Four teams of roughly three engineers each
  - Three teams (AE1–AE3) used the declarative approach
  - Fourth team (TF) a modern vendor-supplied object-oriented UI framework
- Each tasked with rewriting a large number of dialogs and palettes

Results: Productivity

- AE1–AE3 teams
  - completed roughly 75 dialogs and palettes
  - 50 more under way
- TF team
  - completed fewer than 10 altogether

Results: Defect Count



Conclusion

- Programming event-handlers manually is very difficult
- Unrealistic to hope for correct, responsive, feature-rich user interfaces

Conclusion

- Programming event-handlers manually is very difficult
- Unrealistic to hope for correct, responsive, feature-rich user interfaces

- Through careful study of commonalities in UI behavior, it is possible to capture user interface behavior as reusable algorithms
- Quality and features can be free