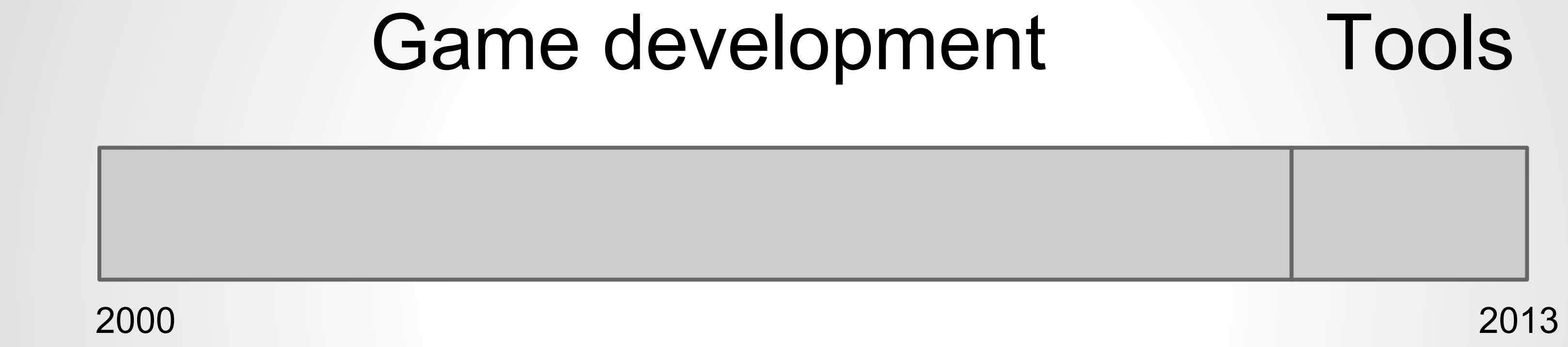
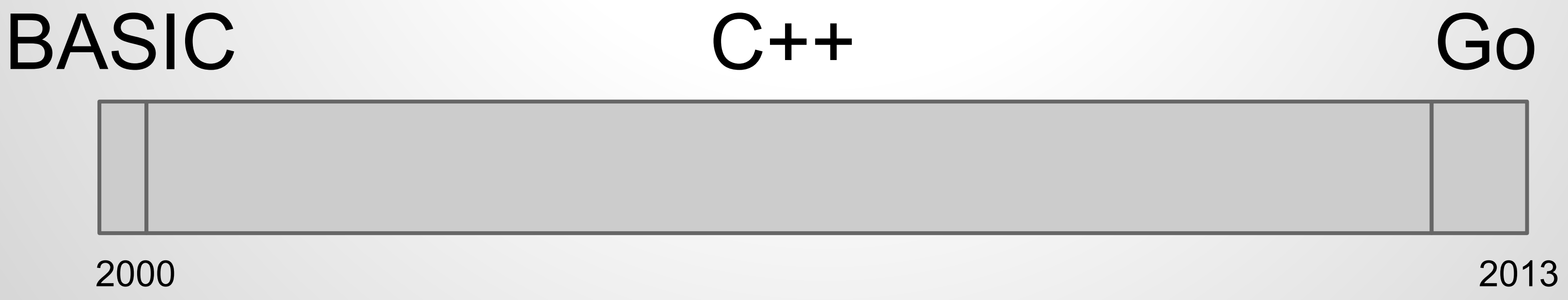
Conception

Presented by Dmitri Shuralyov

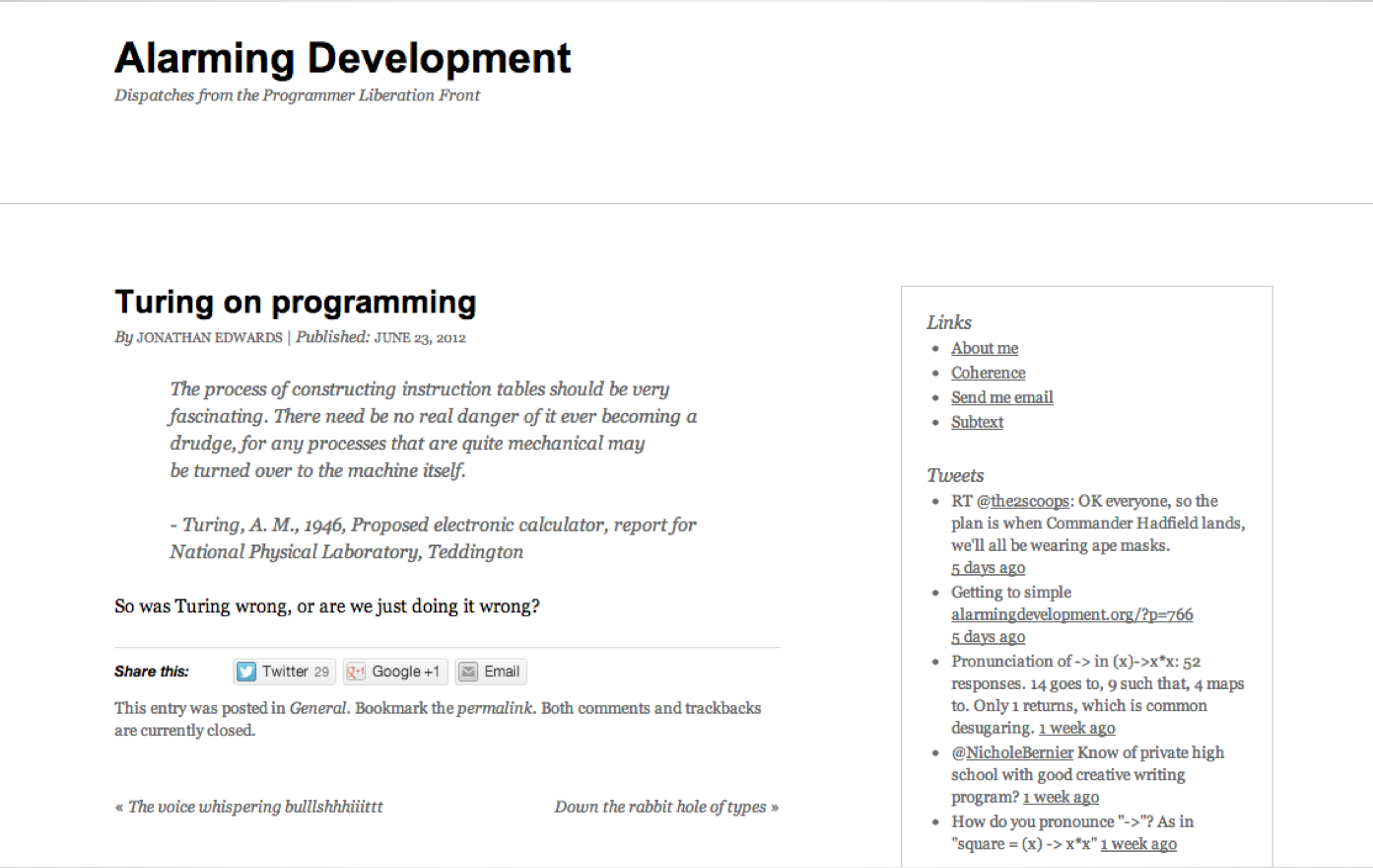
My personal development history





The goals

- Make software development more awesome!
- Faster
  - Why require 3 steps when 1 will do?
    Why require any action at all when 0 steps will do?
- Easier
  - For beginners and pros
- Make software more *soft* (malleable)



Alarming Development

Dispatches from the Programmer Liberation Front

Turning on programming

Turing on programming

By JONATHAN EDWARDS | *Published:* JUNE 23, 2012

> *The process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself.*
>
> *- Turing, A. M., 1946, Proposed electronic calculator, report for National Physical Laboratory, Teddington*

So was Turing wrong, or are we just doing it wrong?

Share this: Twitter 29 | Google +1 | Email

This entry was posted in *General*. Bookmark the *permalink*. Both comments and trackbacks are currently closed.

« The voice whispering bulllshhhiiittt | *Down the rabbit hole of types »*

Links

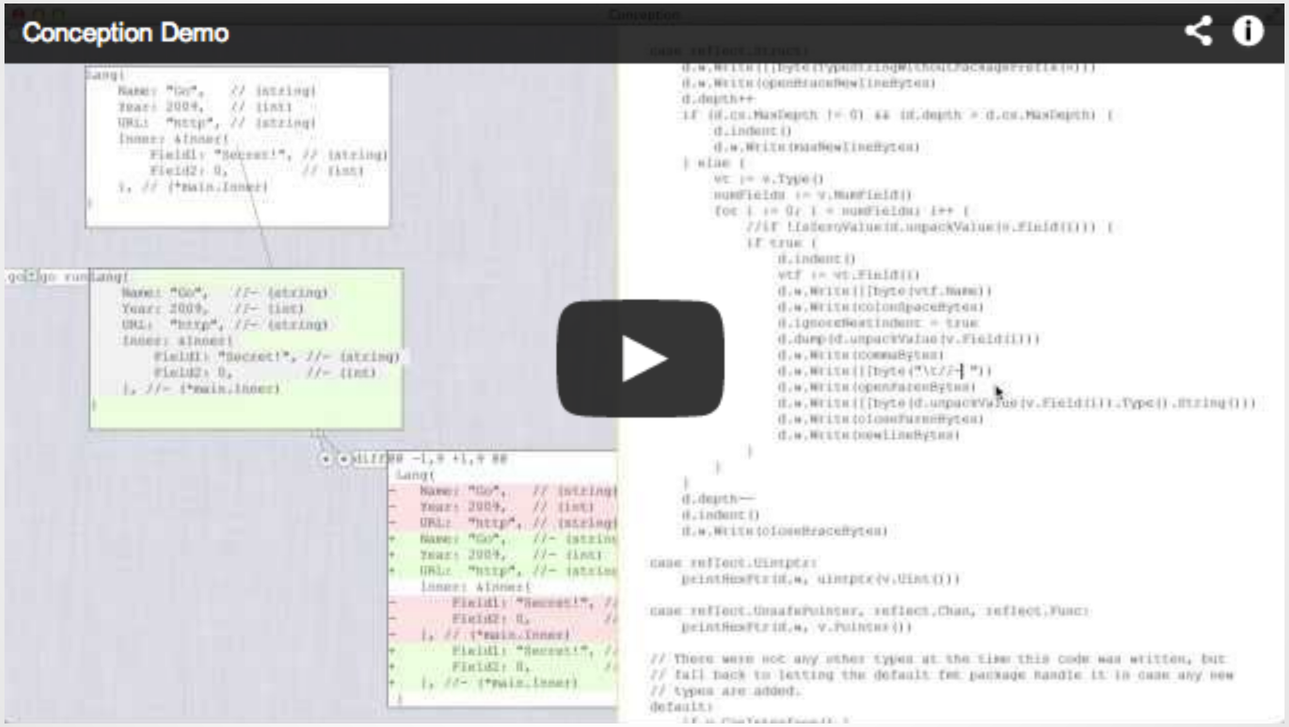
- About me
- Coherence
- Send me email
- Subtext

Tweets

- RT @the2scoops: OK everyone, so the plan is when Commander Hadfield lands, we'll all be wearing ape masks. 5 days ago
- Getting to simple alarmingdevelopment.org/?p=766 5 days ago
- Pronunciation of -> in (x)->x*x: 52 responses. 14 goes to, 9 such that, 4 maps to. Only 1 returns, which is common desugaring. 1 week ago
- @NicholeBernier Know of private high school with good creative writing program? 1 week ago
- How do you pronounce "->"? As in "square = (x) -> x*x" 1 week ago

alarmingdevelopment.org/?p=711

What is Conception?



```
Lang{
    Name: "Go",    // (string)
    Year: 2009,    // (int)
    URL:  "http", // (string)
    Inner: &Inner{
        Field1: "Secret!", // (string)
        Field2: 0,         // (int)
    }, // (*main.Inner)
}
```

```
go run Lang{
    Name: "Go",    //- (string)
    Year: 2009,    //- (int)
    URL:  "http", //- (string)
    Inner: &Inner{
        Field1: "Secret!", //- (string)
        Field2: 0,         //- (int)
    }, //- (*main.Inner)
}
```

```
diff @@ -1,9 +1,9 @@
 Lang{
-    Name: "Go",    // (string)
-    Year: 2009,    // (int)
-    URL:  "http", // (string)
+    Name: "Go",    //- (strin
+    Year: 2009,    //- (int)
+    URL:  "http", //- (strin
     Inner: &Inner{
-        Field1: "Secret!", //
-        Field2: 0,         //
-    }, // (*main.Inner)
+        Field1: "Secret!", //
+        Field2: 0,         //
+    }, //- (*main.Inner)
 }
```

```
case reflect.Struct:
    d.w.Write([]byte(typeStringWithoutPackagePrefix(v)))
    d.w.Write(openBraceNewlineBytes)
    d.depth++
    if (d.cs.MaxDepth != 0) && (d.depth > d.cs.MaxDepth) {
        d.indent()
        d.w.Write(maxNewlineBytes)
    } else {
        vt := v.Type()
        numFields := v.NumField()
        for i := 0; i < numFields; i++ {
            //if !isZeroValue(d.unpackValue(v.Field(i))) {
            if true {
                d.indent()
                vtf := vt.Field(i)
                d.w.Write([]byte(vtf.Name))
                d.w.Write(colonSpaceBytes)
                d.ignoreNextIndent = true
                d.dump(d.unpackValue(v.Field(i)))
                d.w.Write(commaBytes)
                d.w.Write([]byte("\t//- "))
                d.w.Write(openParenBytes)
                d.w.Write([]byte(d.unpackValue(v.Field(i)).Type().String()))
                d.w.Write(closeParenBytes)
                d.w.Write(newlineBytes)
            }
        }
    }
    d.depth--
    d.indent()
    d.w.Write(closeBraceBytes)

case reflect.Uintptr:
    printHexPtr(d.w, uintptr(v.Uint()))

case reflect.UnsafePointer, reflect.Chan, reflect.Func:
    printHexPtr(d.w, v.Pointer())

// There were not any other types at the time this code was written, but
// fall back to letting the default fmt package handle it in case any new
// types are added.
default:
```

github.com/shurcooL/Conception#demonstration

Design vs. Implementation

- Design
  - The main thing that matters
  - It specifies the look, feel, behaviour of your app

- Implementation
  - It makes your design run on the computer
  - It *shouldn't* be important nor hard

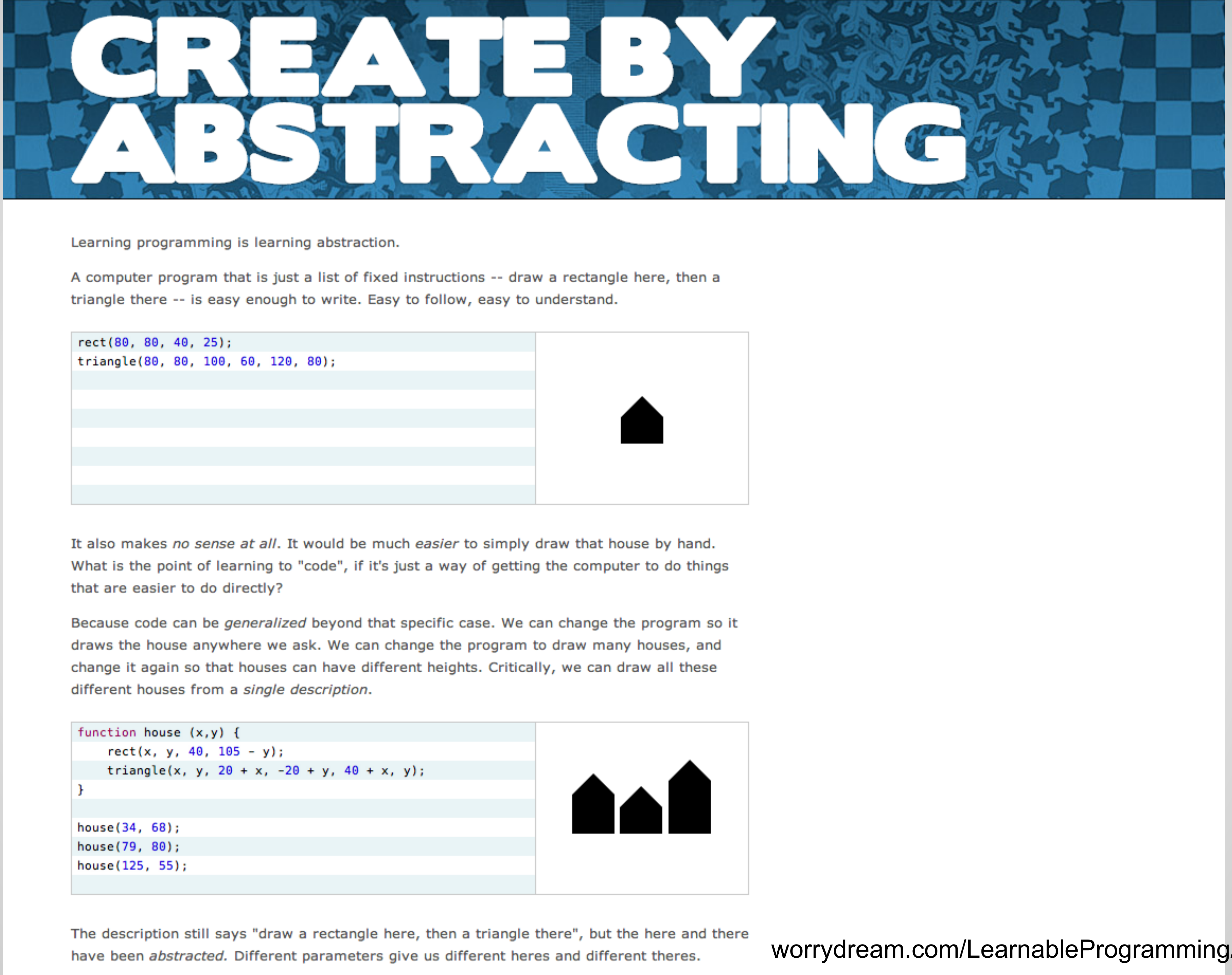
What Conception *really* is

- An evolving **set of guiding principles** that I believe will get us closer to the goal
    - One key guiding principle

- Implementing Conception is about finding out whether certain ideas work or not

What's great about <u>soft</u>ware?

- Ability to create
- Capacity and ease of change
- Any downside is an opportunity to improve

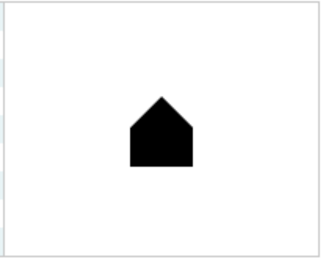


CREATE BY ABSTRACTING

Learning programming is learning abstraction.

A computer program that is just a list of fixed instructions -- draw a rectangle here, then a triangle there -- is easy enough to write. Easy to follow, easy to understand.

```
rect(80, 80, 40, 25);
triangle(80, 80, 100, 60, 120, 80);
```



It also makes *no sense at all*. It would be much *easier* to simply draw that house by hand. What is the point of learning to "code", if it's just a way of getting the computer to do things that are easier to do directly?

Because code can be *generalized* beyond that specific case. We can change the program so it draws the house anywhere we ask. We can change the program to draw many houses, and change it again so that houses can have different heights. Critically, we can draw all these different houses from a *single description*.

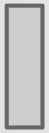
```
function house (x,y) {
    rect(x, y, 40, 105 - y);
    triangle(x, y, 20 + x, -20 + y, 40 + x, y);
}

house(34, 68);
house(79, 80);
house(125, 55);
```

The description still says "draw a rectangle here, then a triangle there", but the here and there have been *abstracted*. Different parameters give us different heres and different theres.

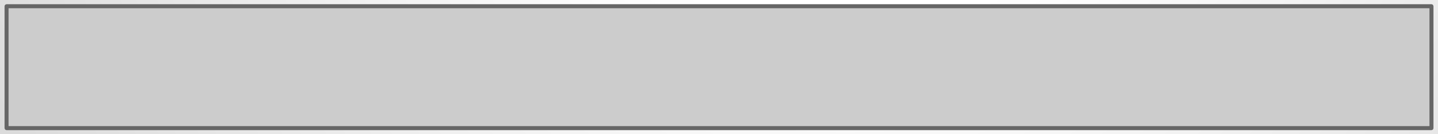
worrydream.com/LearnableProgramming/

Blank slate

Progress

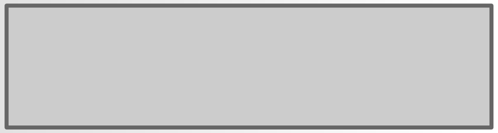


Capacity for change

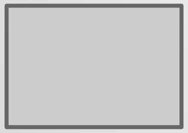


After rapid prototyping

Progress

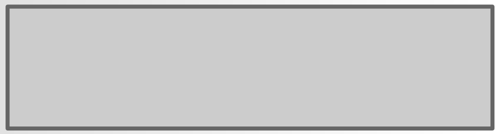


Capacity for change

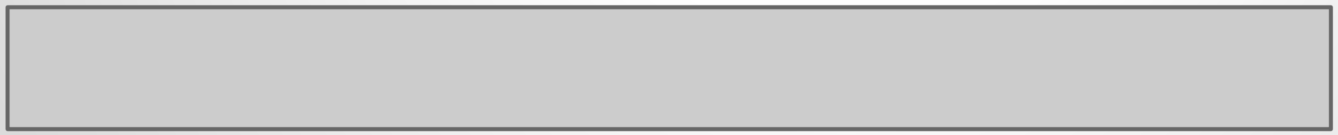


Enter refactoring!

Progress



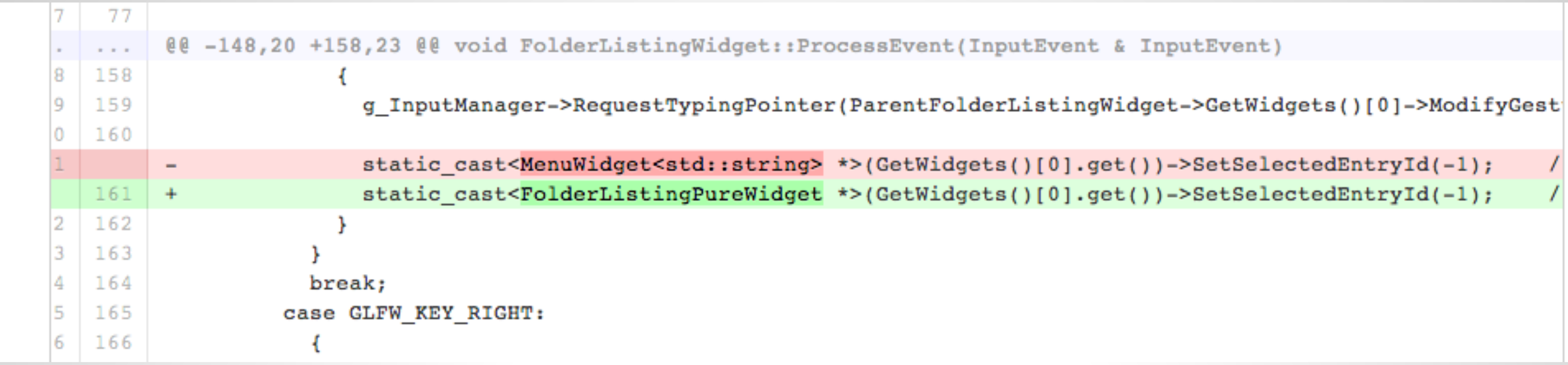
Capacity for change



A hurdle to tackle

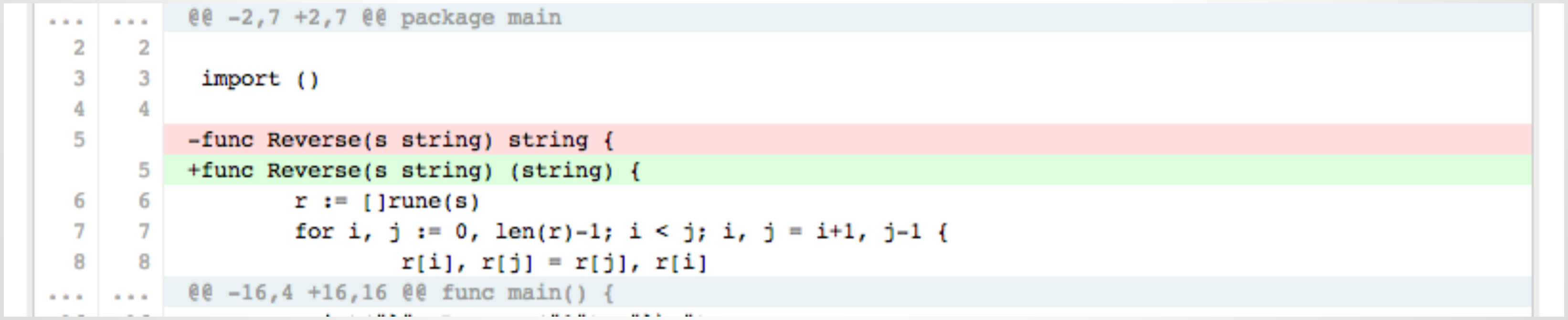
- Imagine you have a function that does the same thing in multiple projects

- You decide to improve it



```
@@ -148,20 +158,23 @@ void FolderListingWidget::ProcessEvent(InputEvent & InputEvent)
              {
                g_InputManager->RequestTypingPointer(ParentFolderListingWidget->GetWidgets()[0]->ModifyGest

-               static_cast<MenuWidget<std::string> *>(GetWidgets()[0].get())->SetSelectedEntryId(-1);    /
+               static_cast<FolderListingPureWidget *>(GetWidgets()[0].get())->SetSelectedEntryId(-1);    /
              }
            }
            break;
          case GLFW_KEY_RIGHT:
            {
```

github.com



```
@@ -2,7 +2,7 @@ package main

 import ()

-func Reverse(s string) string {
+func Reverse(s string) (string) {
 	r := []rune(s)
 	for i, j := 0, len(r)-1; i < j; i, j = i+1, j-1 {
 		r[i], r[j] = r[j], r[i]
@@ -16,4 +16,16 @@ func main() {
```

gist.github.com

Code duplication is bad

- To improve X, you have to improve it in multiple places

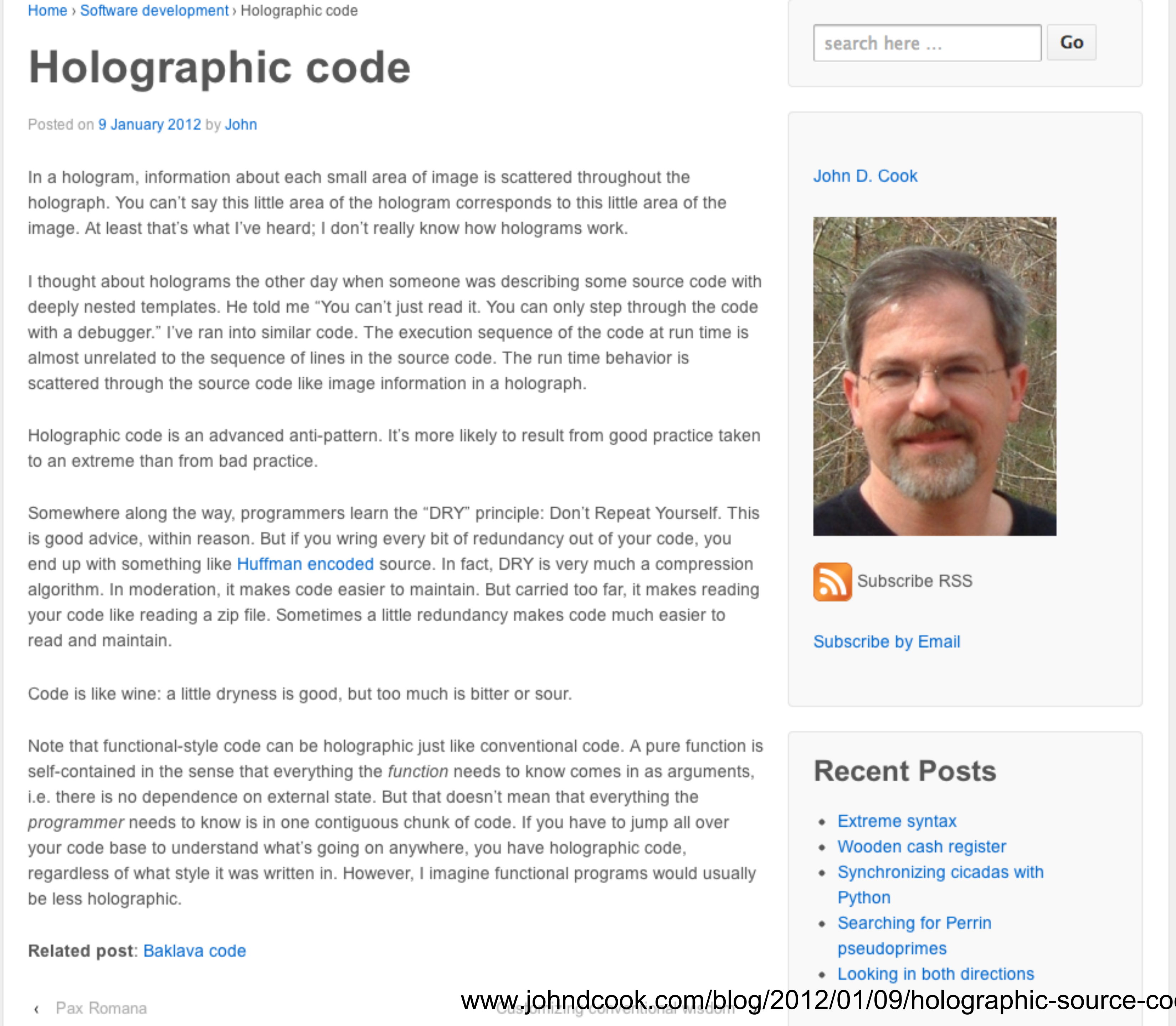
- If you forget to change all instances of X, you'll create inconsistency bugs

- When you see duplicated code, you will not even want to touch it, so it will remain unimproved

DRY

Don't Repeat Yourself

- Duplication of efforts
  - I really dislike having to do the same work more than once
- Duplication of code
  - I really dislike having to manually change the value of one decision in more than one place
- (*Automatic* duplication is fine: backups, cache)



Home › Software development › Holographic code

Holographic code

Posted on 9 January 2012 by John

In a hologram, information about each small area of image is scattered throughout the holograph. You can't say this little area of the hologram corresponds to this little area of the image. At least that's what I've heard; I don't really know how holograms work.

I thought about holograms the other day when someone was describing some source code with deeply nested templates. He told me "You can't just read it. You can only step through the code with a debugger." I've ran into similar code. The execution sequence of the code at run time is almost unrelated to the sequence of lines in the source code. The run time behavior is scattered through the source code like image information in a holograph.

Holographic code is an advanced anti-pattern. It's more likely to result from good practice taken to an extreme than from bad practice.

Somewhere along the way, programmers learn the "DRY" principle: Don't Repeat Yourself. This is good advice, within reason. But if you wring every bit of redundancy out of your code, you end up with something like Huffman encoded source. In fact, DRY is very much a compression algorithm. In moderation, it makes code easier to maintain. But carried too far, it makes reading your code like reading a zip file. Sometimes a little redundancy makes code much easier to read and maintain.

Code is like wine: a little dryness is good, but too much is bitter or sour.

Note that functional-style code can be holographic just like conventional code. A pure function is self-contained in the sense that everything the *function* needs to know comes in as arguments, i.e. there is no dependence on external state. But that doesn't mean that everything the *programmer* needs to know is in one contiguous chunk of code. If you have to jump all over your code base to understand what's going on anywhere, you have holographic code, regardless of what style it was written in. However, I imagine functional programs would usually be less holographic.

Related post: Baklava code

‹ Pax Romana   Customizing conventional wisdom

www.johndcook.com/blog/2012/01/09/holographic-source-code/

Tools

- Tools can enable one to work in a fundamentally different way

- Create fundamentally new things

Transformative tools

- Text editors (with copy and paste)
- Version control systems (git)
- Compilers

Insight

- Everything of value is made up of other things of value
- Value can be represented as composition of dependencies
- (There's value in your private helper code, expose it!)

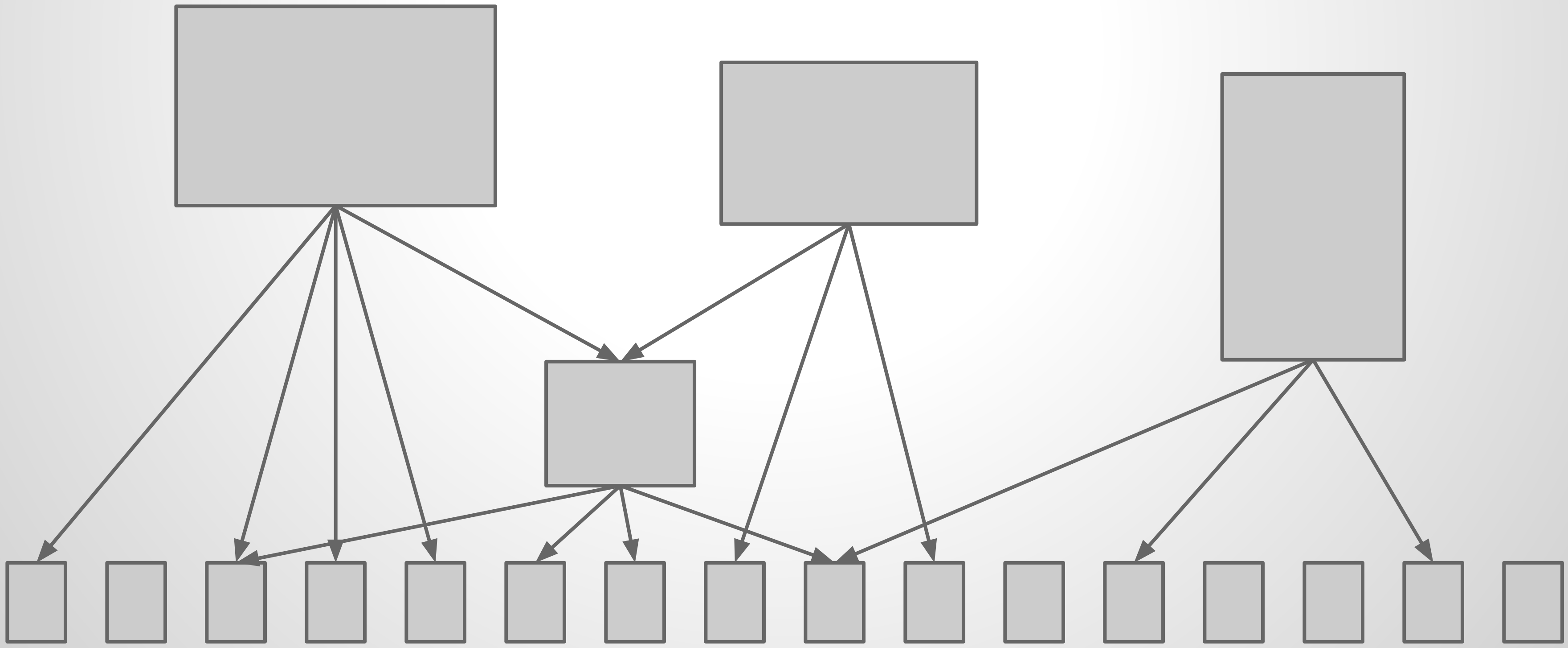
Pure functions

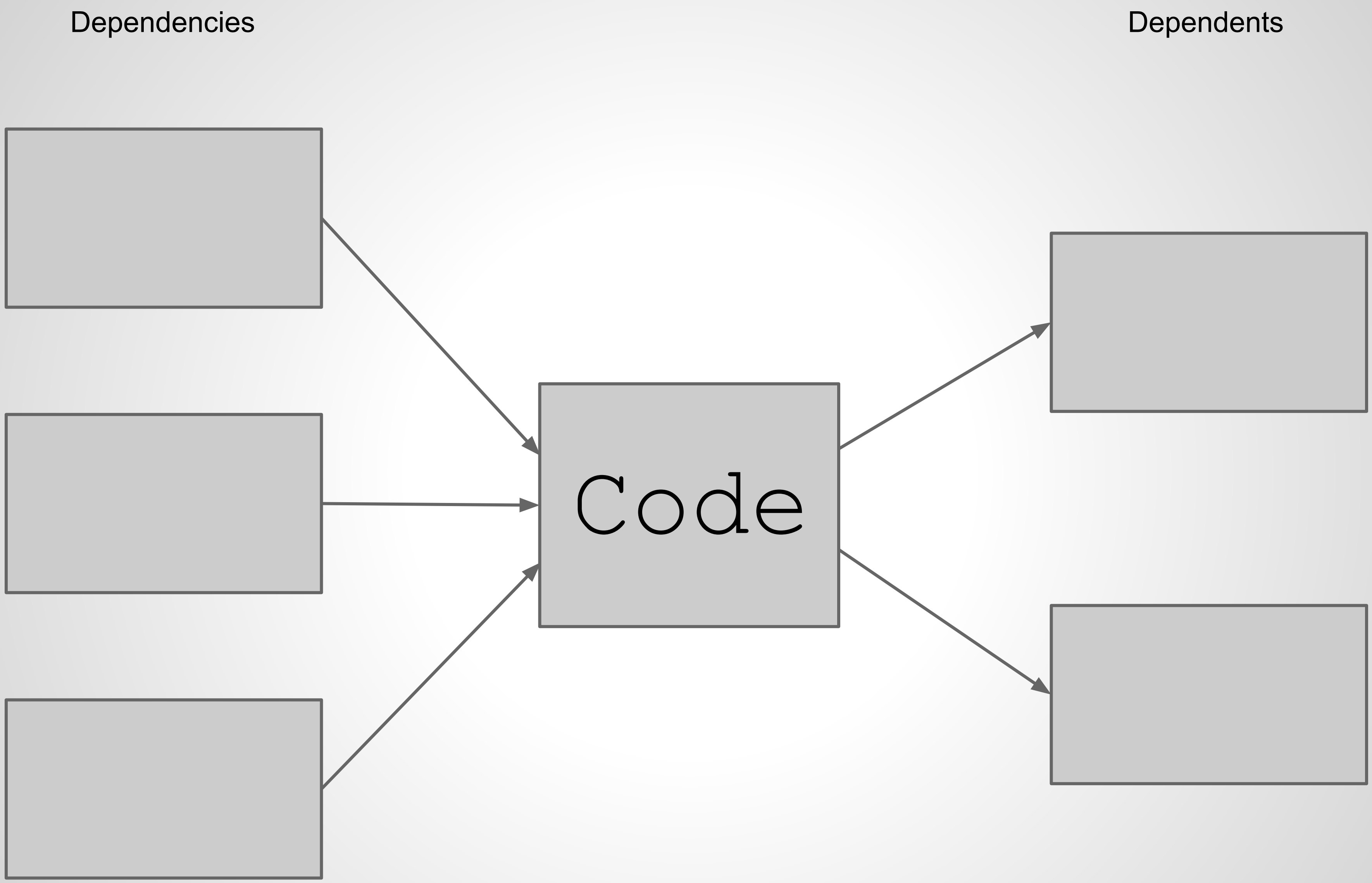
- They make their dependencies and side-effects explicit

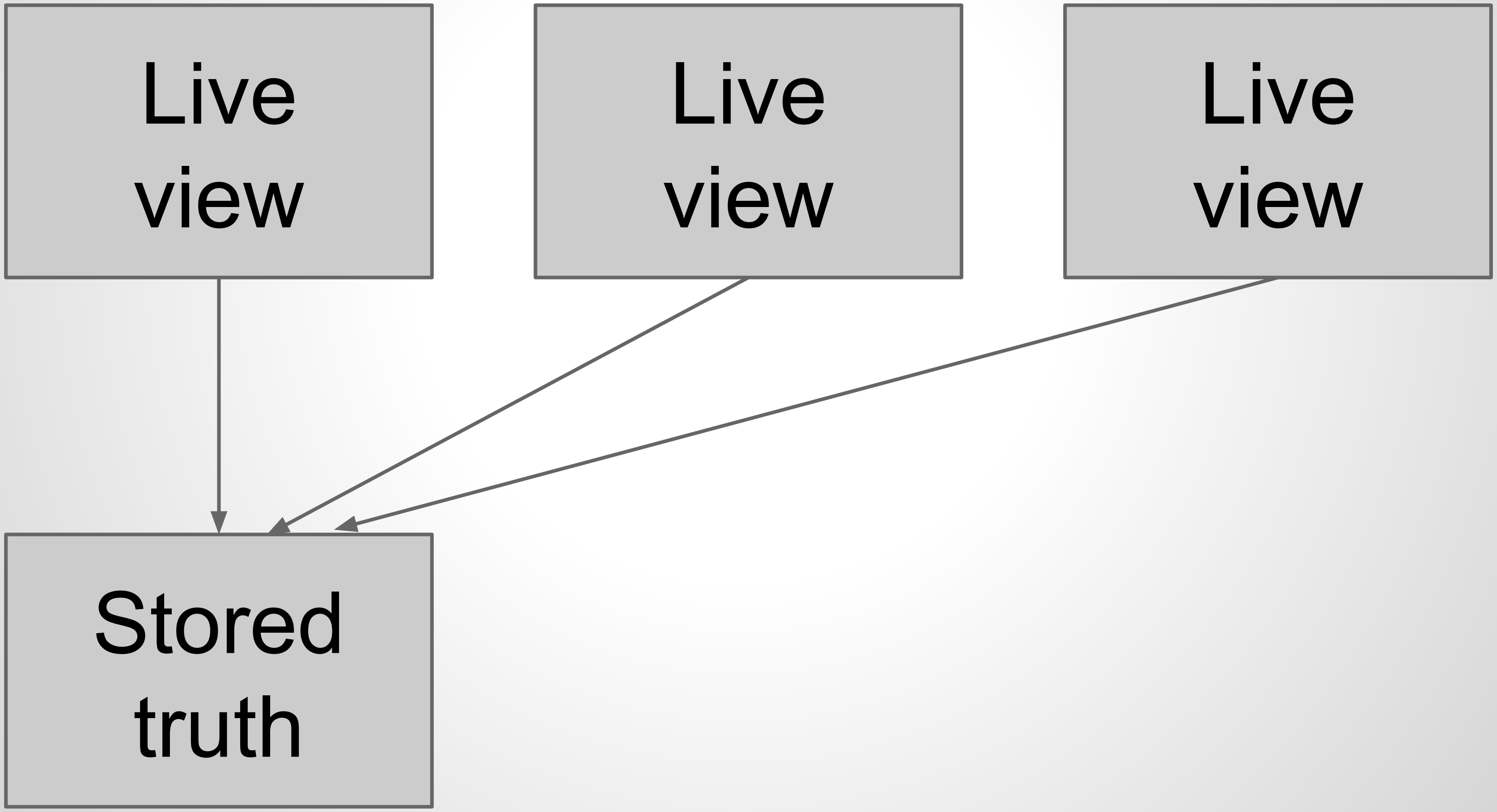
- Any non-pure function can be rewritten as pure

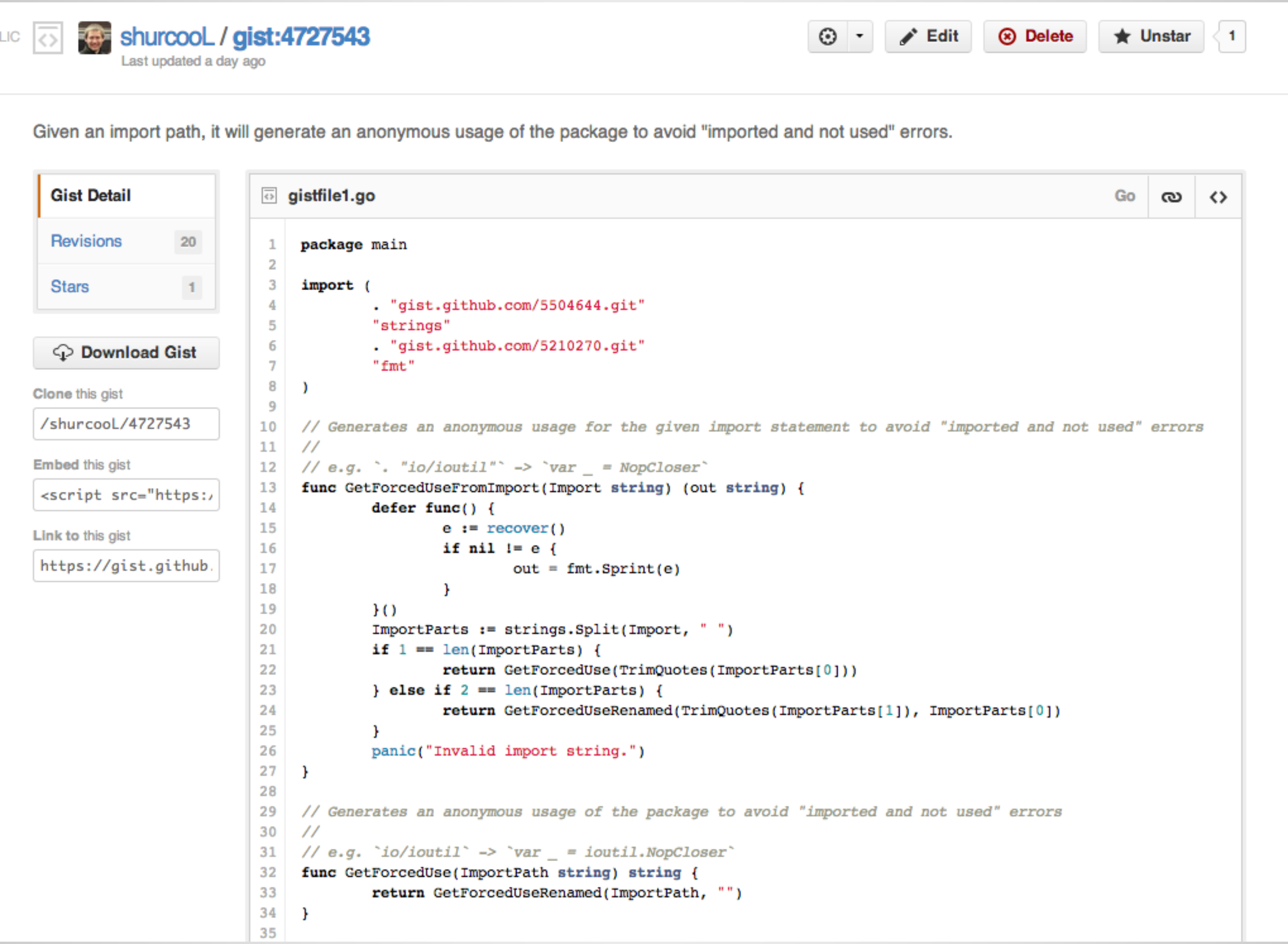
Software as Lego

- Software can be made up of pure functions that are reused across projects



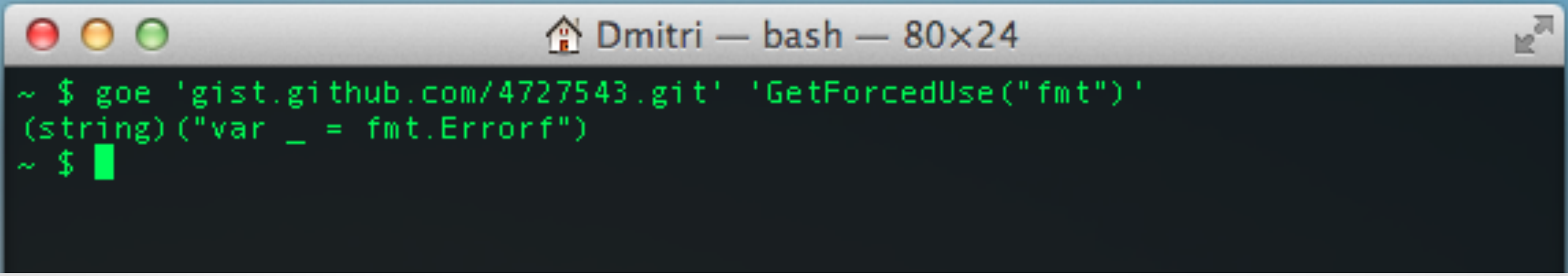

Dependencies
Dependents
Code


Live view
Live view
Live view
Stored truth

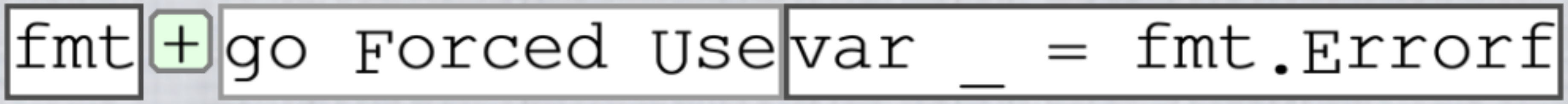


Given an import path, it will generate an anonymous usage of the package to avoid "imported and not used" errors.

```go
package main

import (
	. "gist.github.com/5504644.git"
	"strings"
	. "gist.github.com/5210270.git"
	"fmt"
)

// Generates an anonymous usage for the given import statement to avoid "imported and not used" errors
//
// e.g. `. "io/ioutil"` -> `var _ = NopCloser`
func GetForcedUseFromImport(Import string) (out string) {
	defer func() {
		e := recover()
		if nil != e {
			out = fmt.Sprint(e)
		}
	}()
	ImportParts := strings.Split(Import, " ")
	if 1 == len(ImportParts) {
		return GetForcedUse(TrimQuotes(ImportParts[0]))
	} else if 2 == len(ImportParts) {
		return GetForcedUseRenamed(TrimQuotes(ImportParts[1]), ImportParts[0])
	}
	panic("Invalid import string.")
}

// Generates an anonymous usage of the package to avoid "imported and not used" errors
//
// e.g. `io/ioutil` -> `var _ = ioutil.NopCloser`
func GetForcedUse(ImportPath string) string {
	return GetForcedUseRenamed(ImportPath, "")
}

```



```
~ $ goe 'gist.github.com/4727543.git' 'GetForcedUse("fmt")'
(string)("var _ = fmt.Errorf")
~ $ 
```



```
fmt + go Forced Use var _ = fmt.Errorf
```

The dream

- Wikipedia-like home for pure functions; software uses them
- Improve any function and you're making all software that relies on it (directly or indirectly) better
- (Make a function worse, and there's no harm to others)

Thank you!

github.com/shurcooL/Conception

Dmitri Shuralyov
twitter.com/shurcooL
github.com/shurcooL
shurcooL@gmail.com