TRU-TEST GROUP 50 YEARS

THE FUTURE OF FARMING SINCE 1964

What does good software look like?

http://www.wordle.net/create

What are the attributes of good software?

http://www.wordle.net/create

What are the attributes of good software?

"Good software is cheerful software: it behaves cheerfully, and it leaves you cheerful, too."

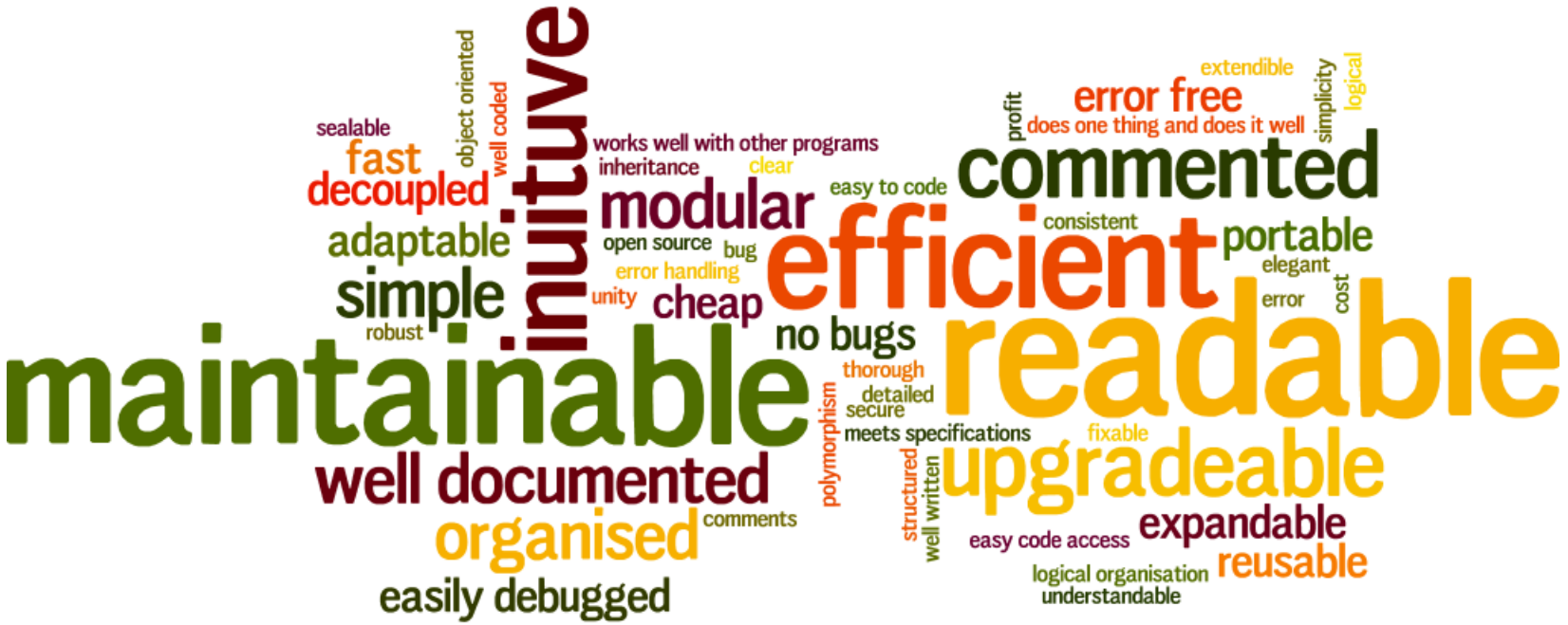
- Freckle Manifesto

inspireUX

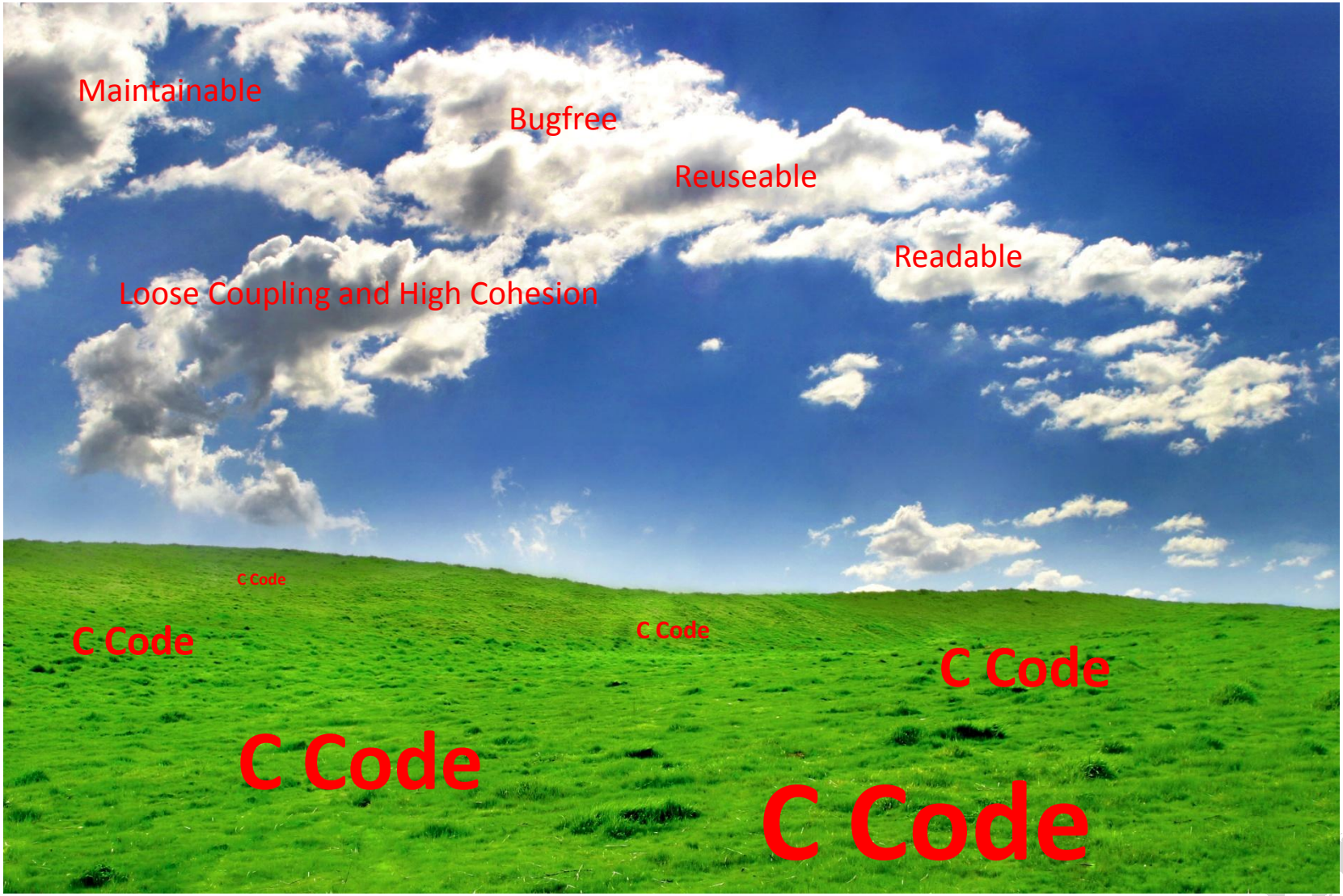
http://www.wordle.net/create

What are the essential attributes of good software?

AFIQ SYAHMI

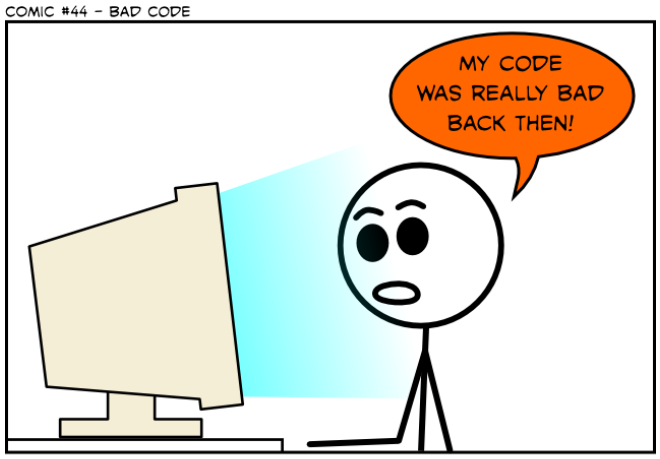


The gap – how do you do it?

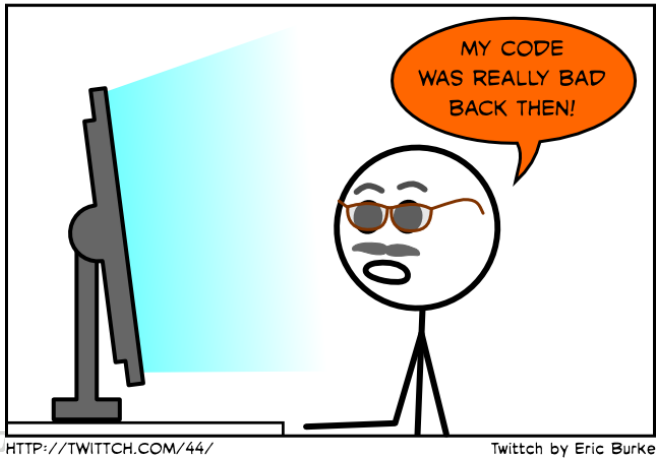


What does bad software look like?

CRAP = Create, Repair, Abandon, rePeat

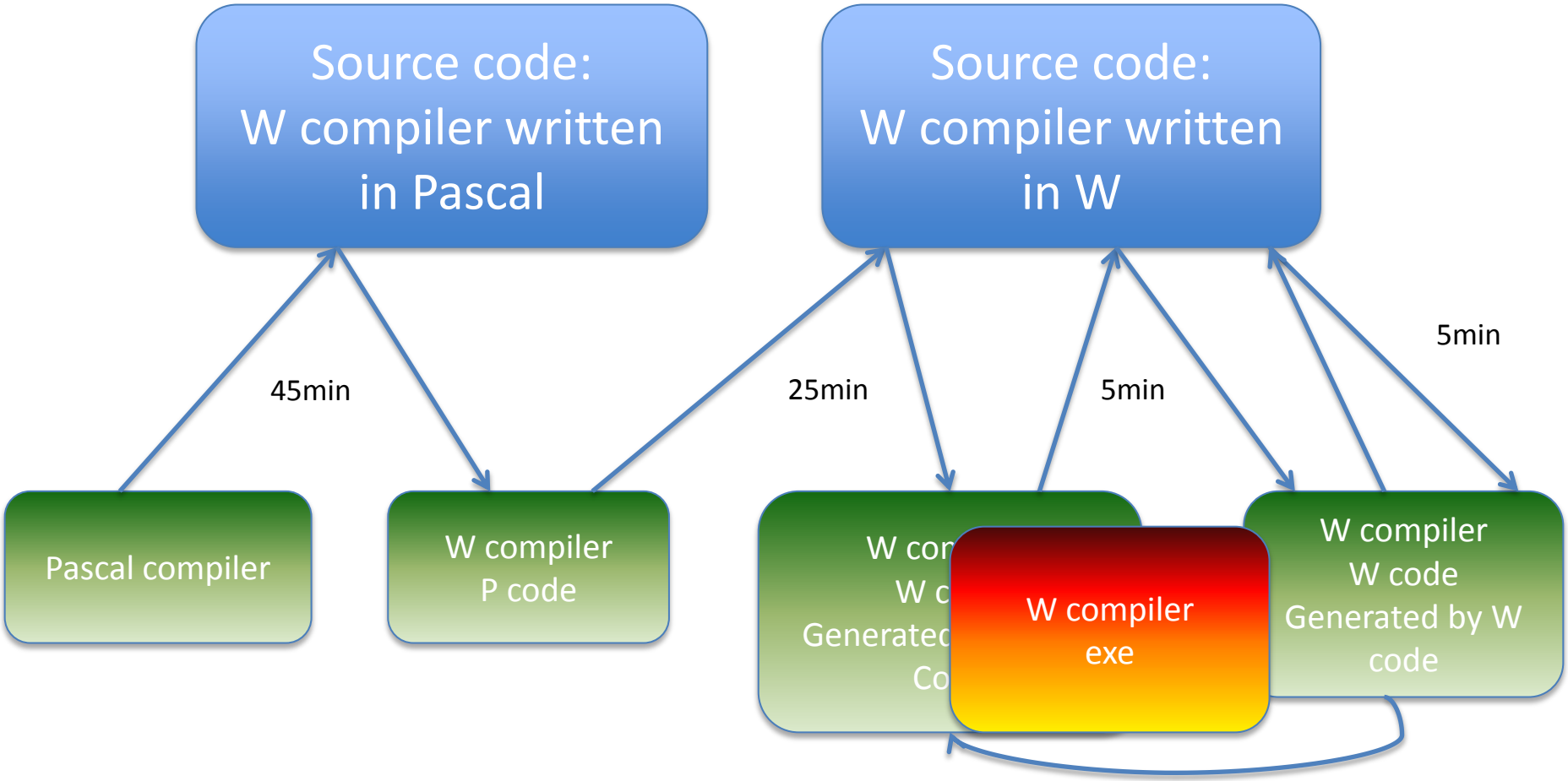


TIME PASSES...



http://www.wordle.net/create

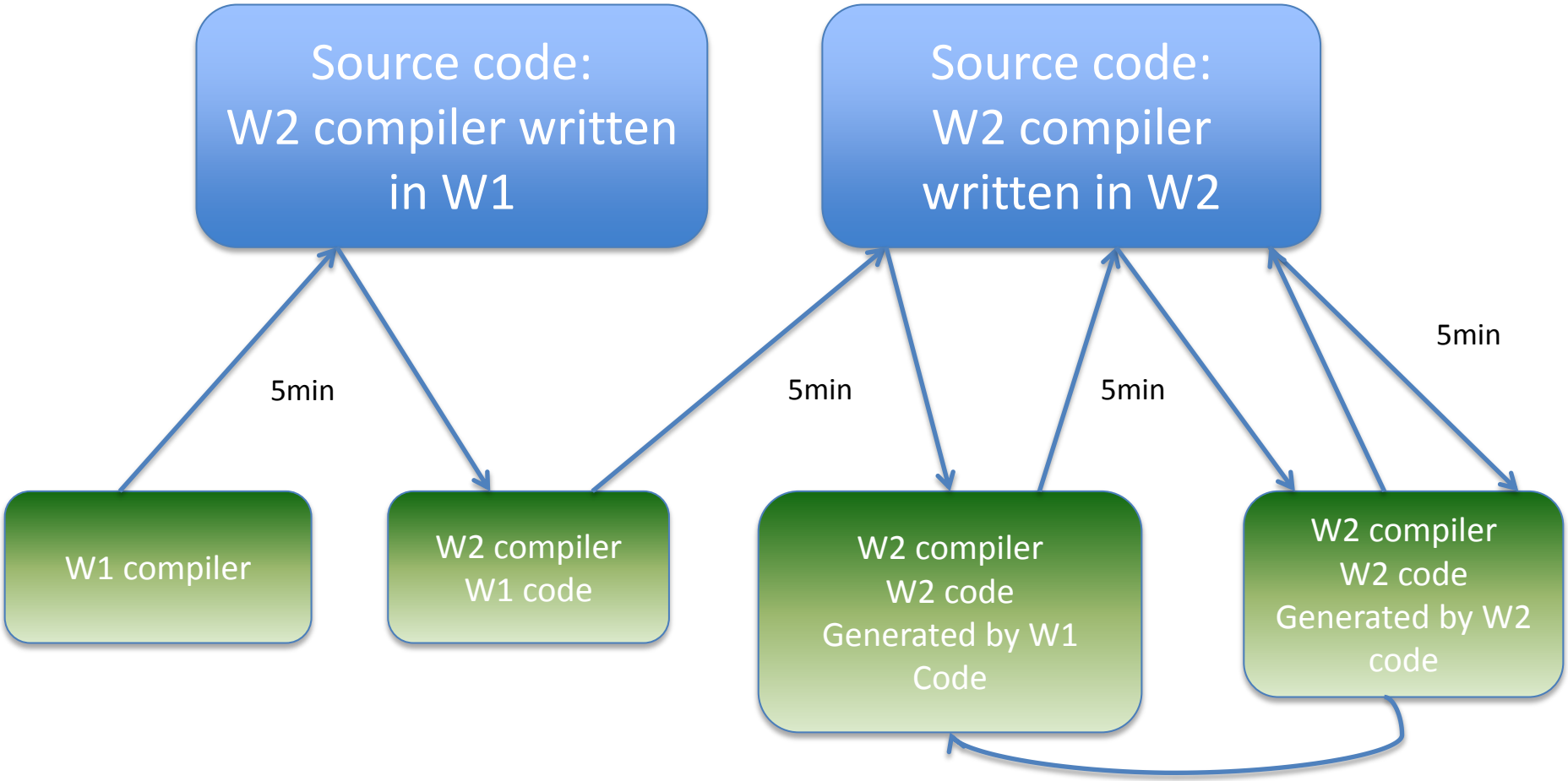
My First big Program – Language ‘W’



Binaries

Identical?

Language ‘W2’



Binaries

Identical?

What does bad software look like?

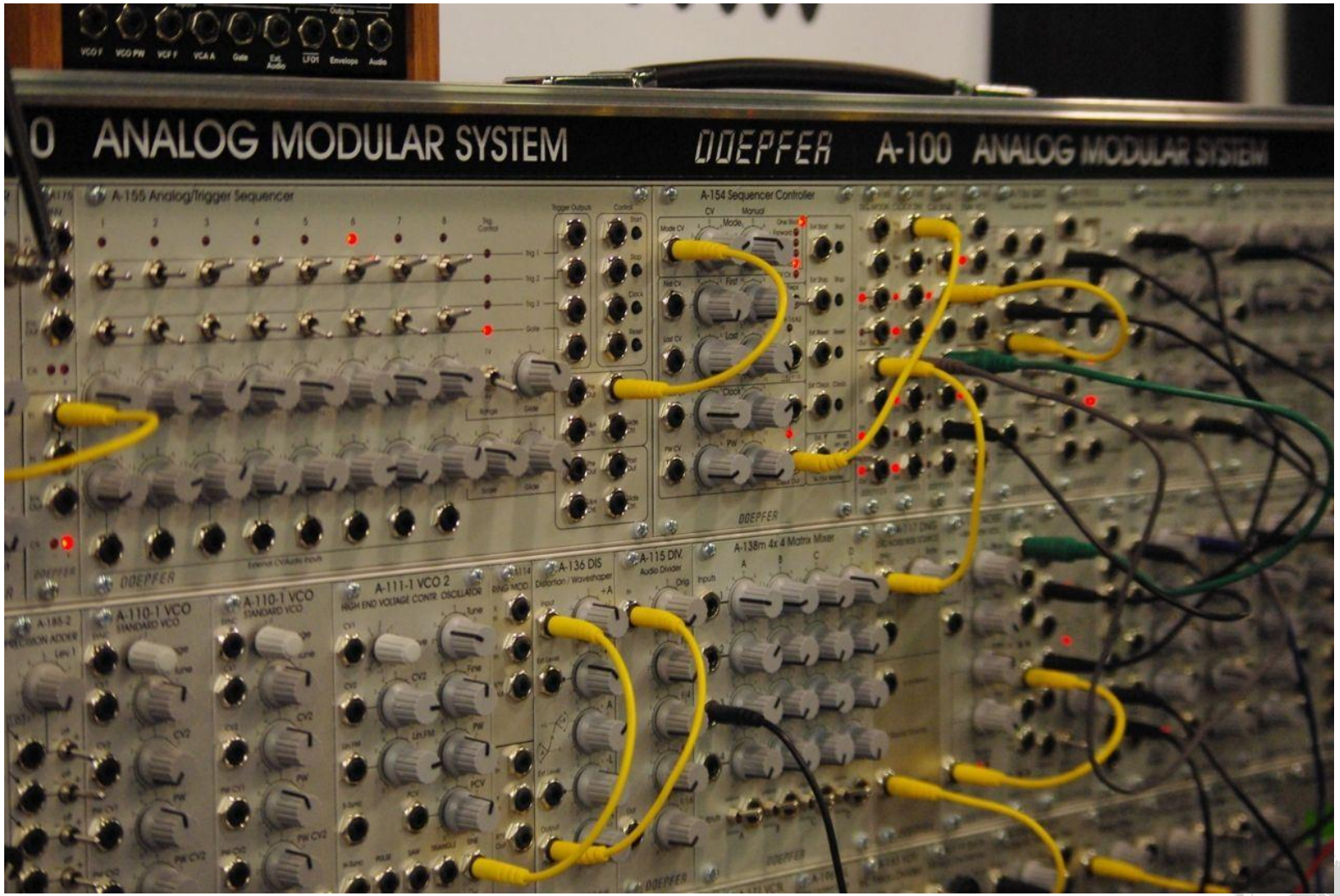
Comments

```
/// Set up the microcontroller port A for controlling the
/// bank load resistors
void SetupMCUforControllingLoadResistors()
{
// See given data sheet page numbers for details
      PortA_DDR = 0xFF; // all outputs, P51
      .
      .
      .
}
```

Meaningful Naming

Names are everywhere in software.
We name our variables, functions, arguments, classes, packages,
source files, directories...
We name and name and name. Because we do so much of it, we better do it well.

Modular/encapsulation



Modules/encapsulation

UserInterface.java

```
protected void startBrewing()
{
   if (hws.isReady() && cv.isReady()) {
      isComplete = false;
      hws.start();
      cv.start();
   }
}
```

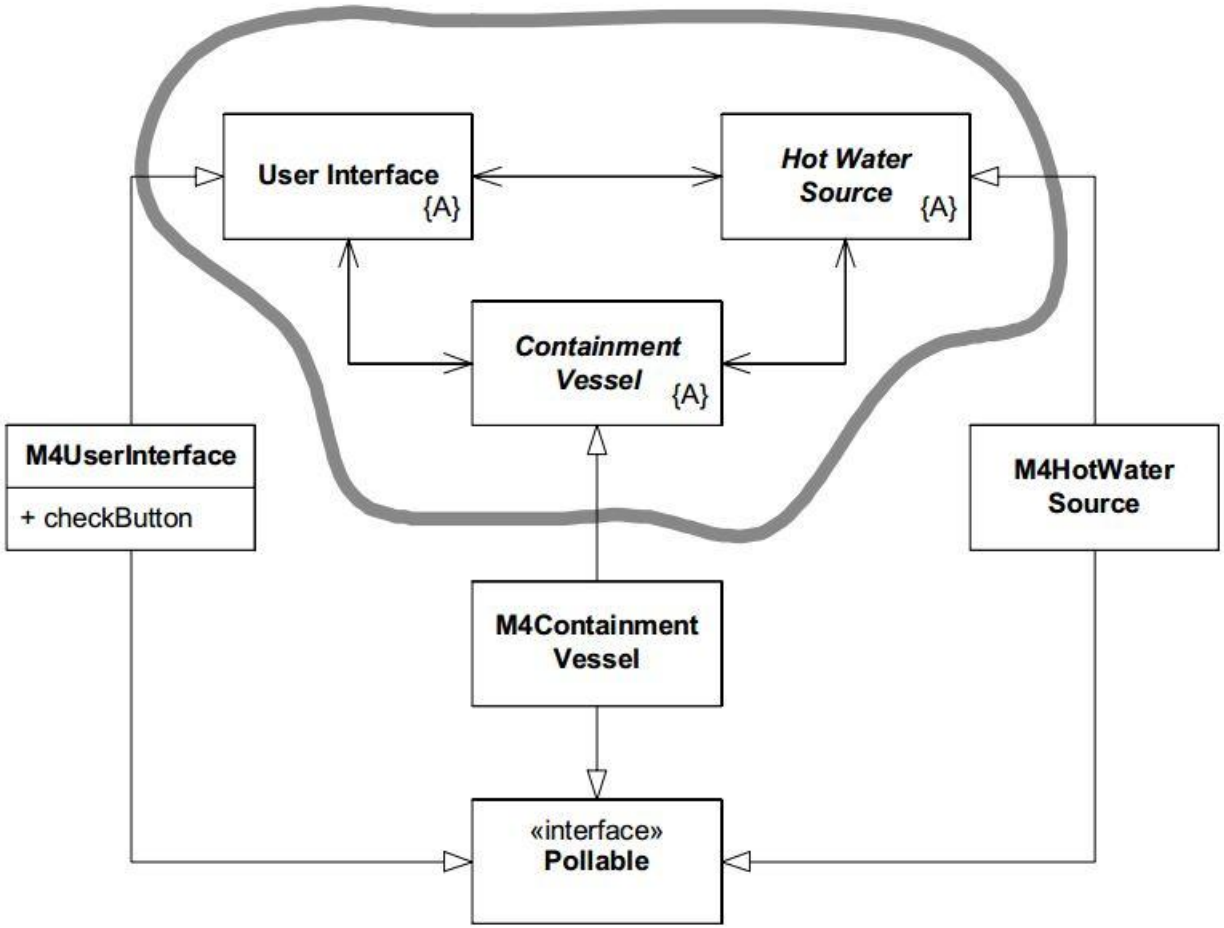
HotWaterSource.java

```
public void start()
{
   isBrewing = true;
   startBrewing();
}

public void done()
{
   isBrewing = false;
}

protected void declareDone()
{
   ui.done();
   cv.done();
   isBrewing = false;
}
```

ContainmentVessel.jav

```
public void start()
{
   isBrewing = true;
   isComplete = false;
}

public void done()
{
   isBrewing = false;
}
protected void declareComplete()
{
   isComplete = true;
   ui.complete();
}
```

OOD/Design patterns



Big pile of Ravioli

Big ball of mud

XR3000

```
/*
Filename:   DRFTSETU.C

Module purpose:
   Draft Setup Page.
            Like all pages, this page can be a target for a navigation object.
            Define a scrolling table page for up to 9 drafting weight ranges.
            Also has draft settings and softkeys

Copyright (c) 2000 Tru-test Ltd.  All rights reserved.

Written by: GRJ                  Date: 2001-08-20
*/

#include <std.h>
#include "system.h"
#include "debug.h"
#include <assert.h>
#include <string.h>
#include <stdio.h>

#include "scrollp.h"
#include "optionf.h"


// --------------------------- Drafting Descriptor -------------------------------


const OPTIONF_tDescriptor DRFTSETU_DraftingDescriptor =
{
   {
      {
         &OPTIONF_VirtualFunctionTable,      // Pointer to Virtual Function Table
         MESSAGES_DraftingLabel,             // Label "DRAFTING"
         &INFO_Drafting,                                          // INFO page if help key is pressed while in this page
         COLUMN1, LINE1,                     // Position on Page
      },
      "SEDR",                     // Scp Command
      eFont1,                     // tFont
      FALSE,                      // BOOL OutputOnly:1
      TRUE,                       // BOOL Asynchronous output:1
   },
   &FILE_GlobalSettings.Drafting,
   MESSAGES_TickUntickStrings,
   0,                                   // default value
};
```



```
// ------------------------------ Softkeys -----------------------------------
```

XR3000

Drafting Setup FILE : F-6 WEAN-HEIFERS

DRAFTING : ☑ MULTI : X DRAFTING RANGES : 2

DRAFT BY : Weight ICONS : Large Arrows

NO EID TIMEOUT : 0.0 NO EID DIRECTION :

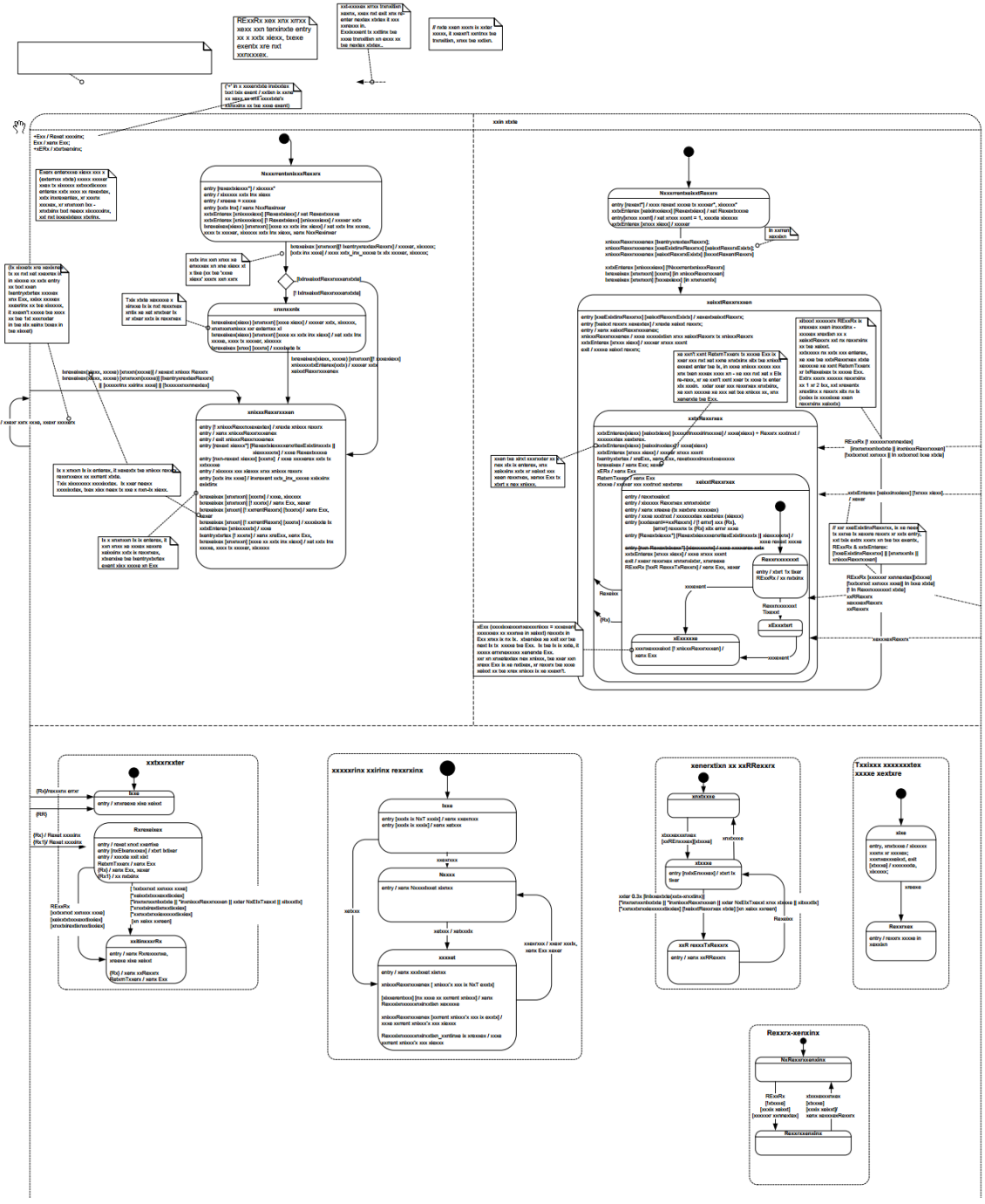
RANGE	FROM	TO	ARROW
1	0.0	100.0	↑
2	100.1	200.0	←

RECALC

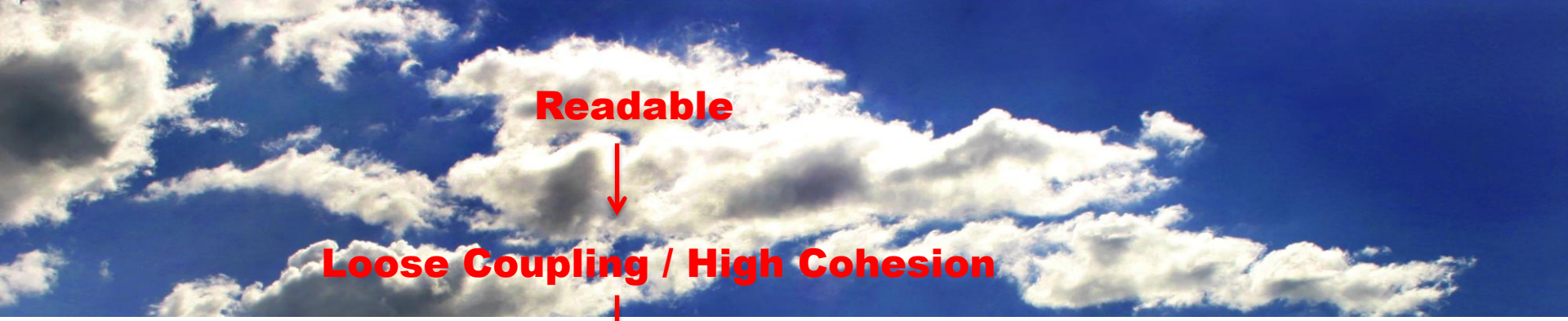
[3000 drftsetu.c](#) [Record.docx](#)

XR5000

[Scale State machine.pdf]

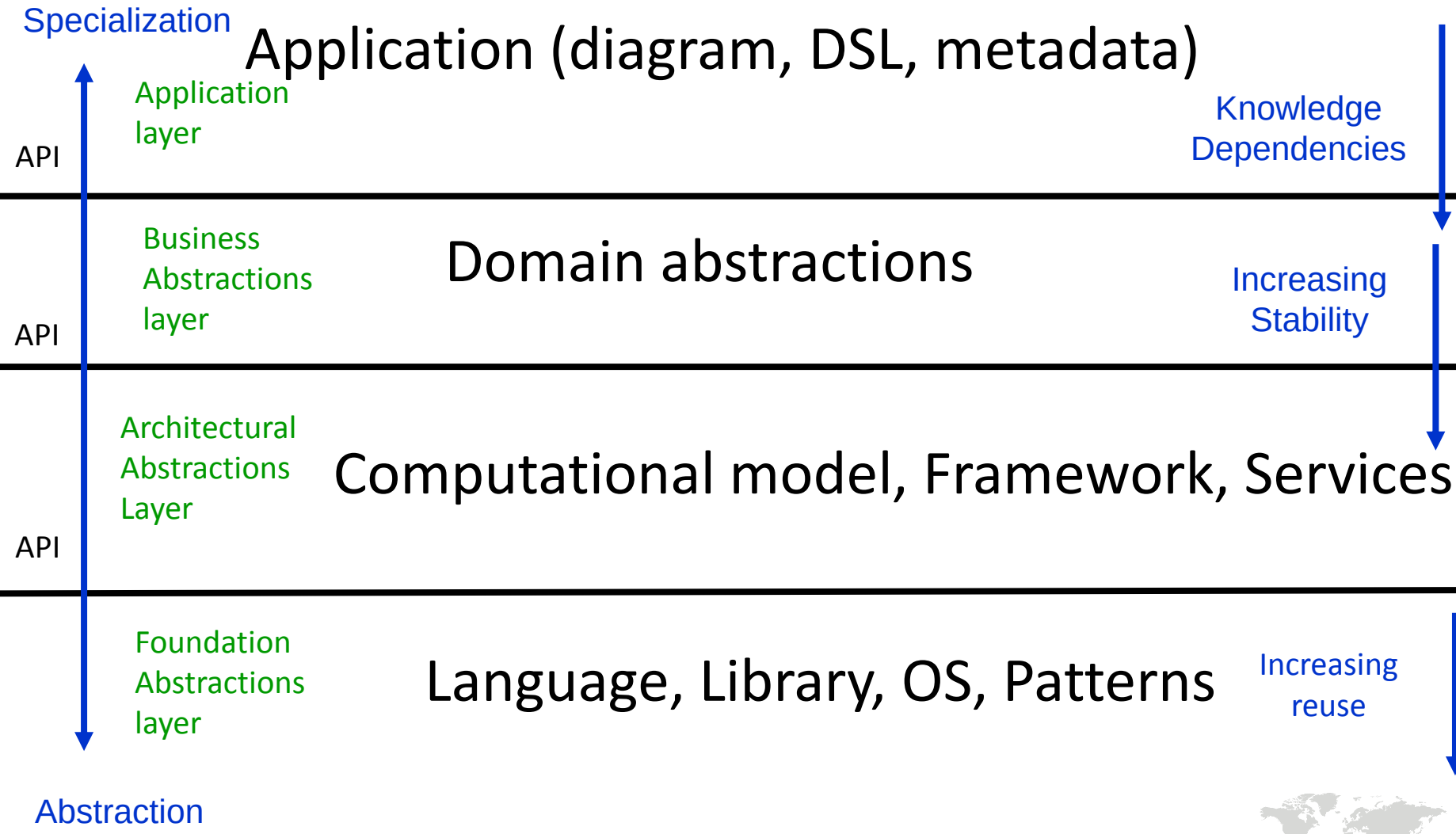


[Scale State machine.pdf]

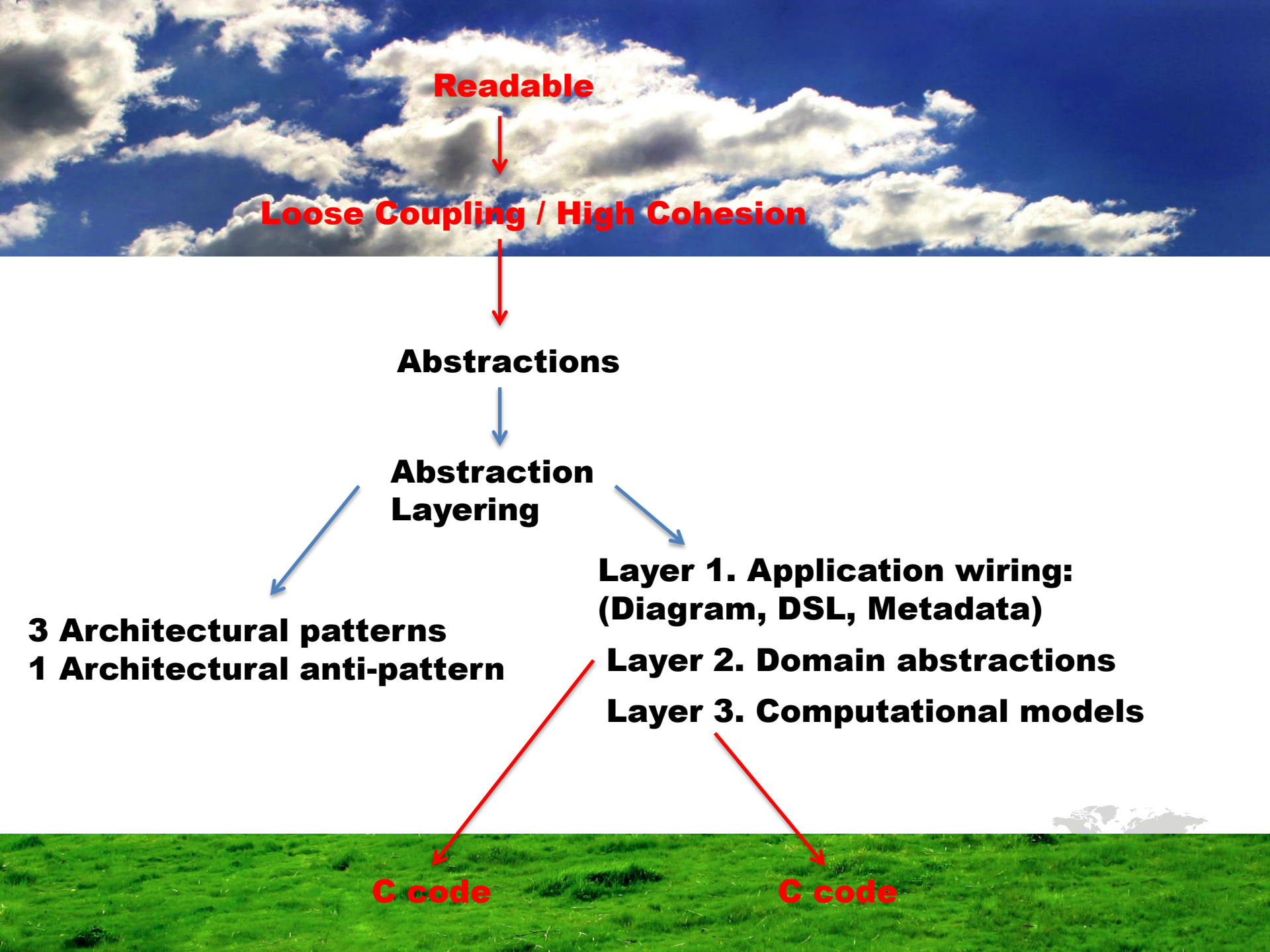


Abstractions

Abstraction Layering



Meilir Page-Jones' abstraction layering


Readable
Loose Coupling / High Cohesion
Abstractions
Abstraction
Layering
3 Architectural patterns
1 Architectural anti-pattern
Layer 1. Application wiring:
(Diagram, DSL, Metadata)
Layer 2. Domain abstractions
Layer 3. Computational models
C code
C code

Coffee machine

Coffee maker – Uncle Bob's problem and solution

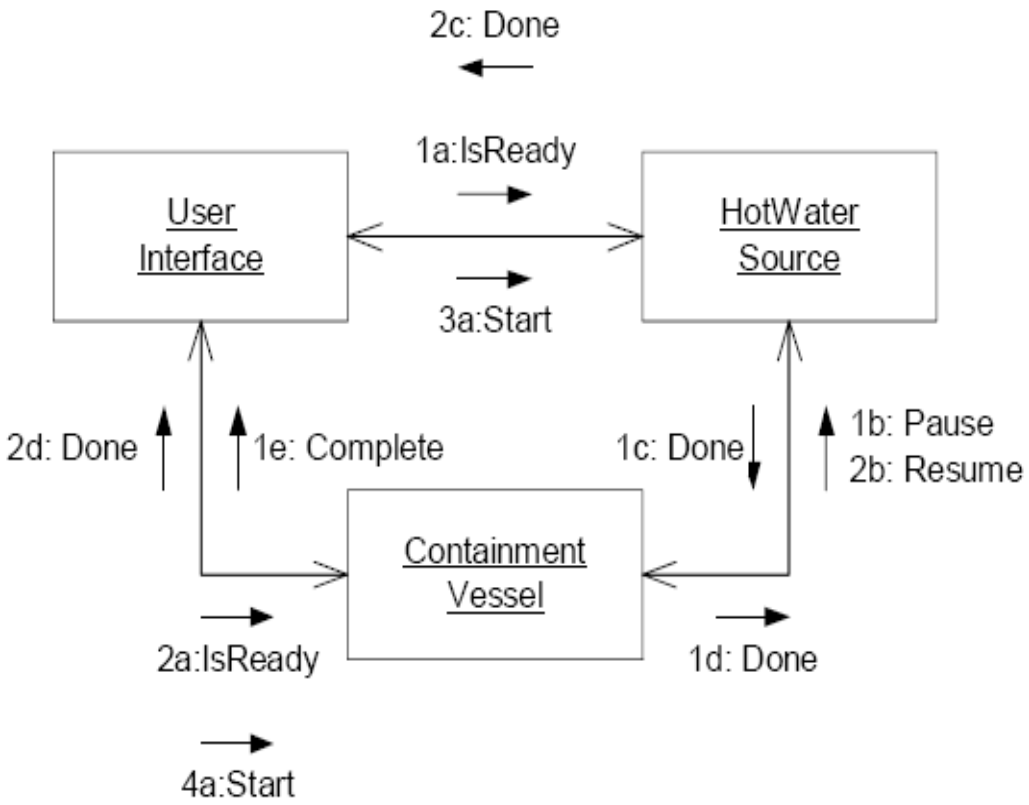


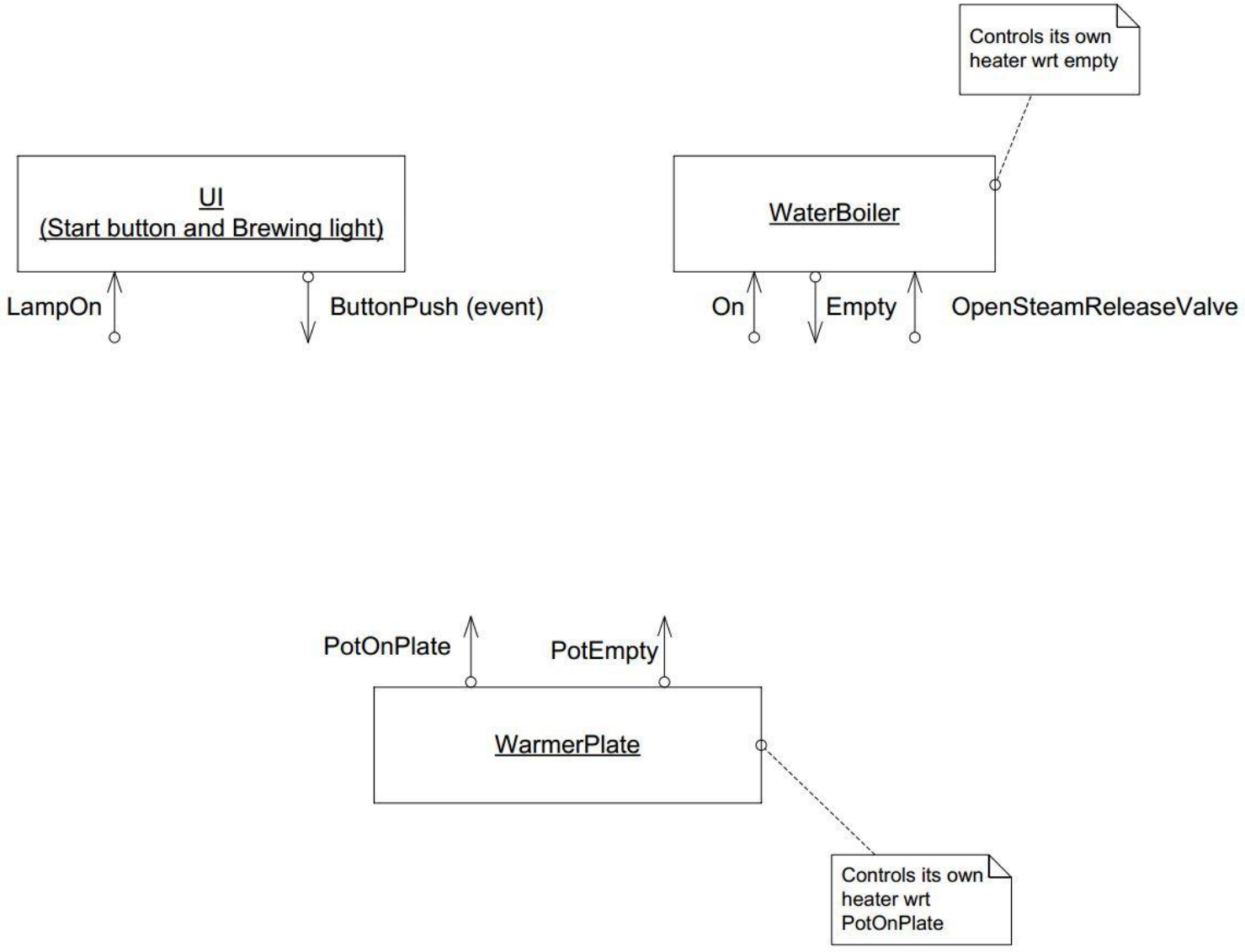
Figure 11–8
Coffee all gone.

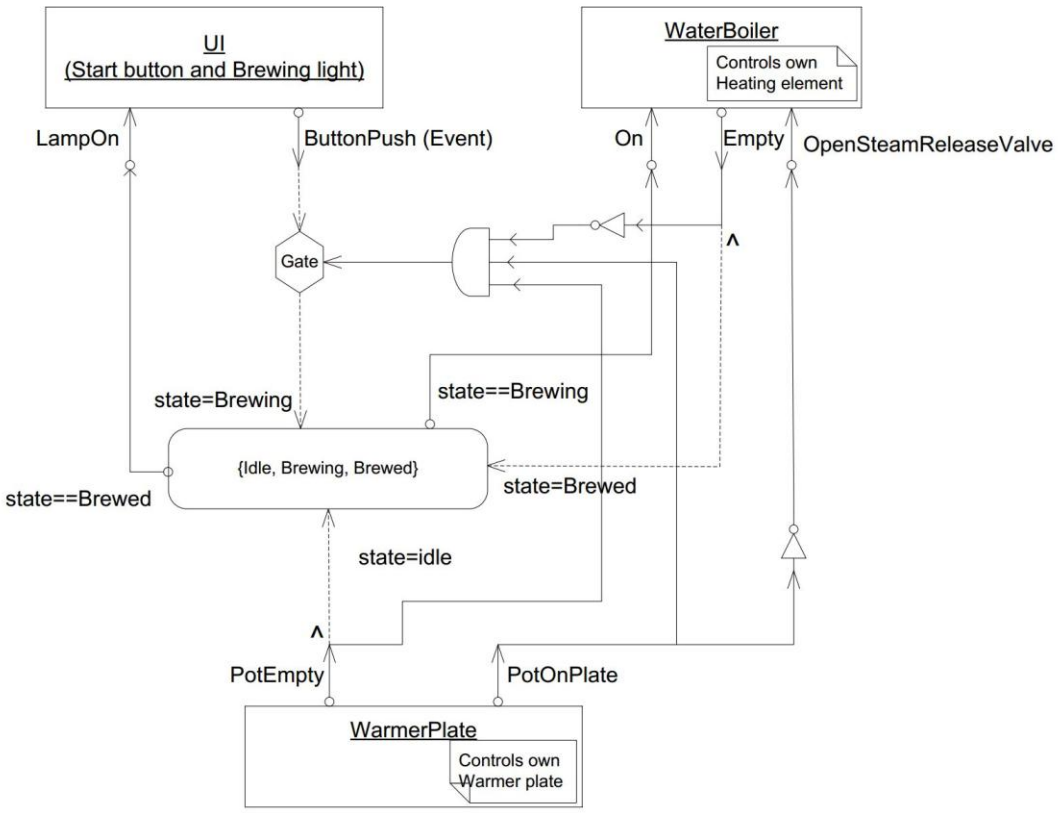
With respect to the abstraction 'Containment Vessel', what do Done, Done, Done, Complete, IsReady, Start, Resume and Pause mean?

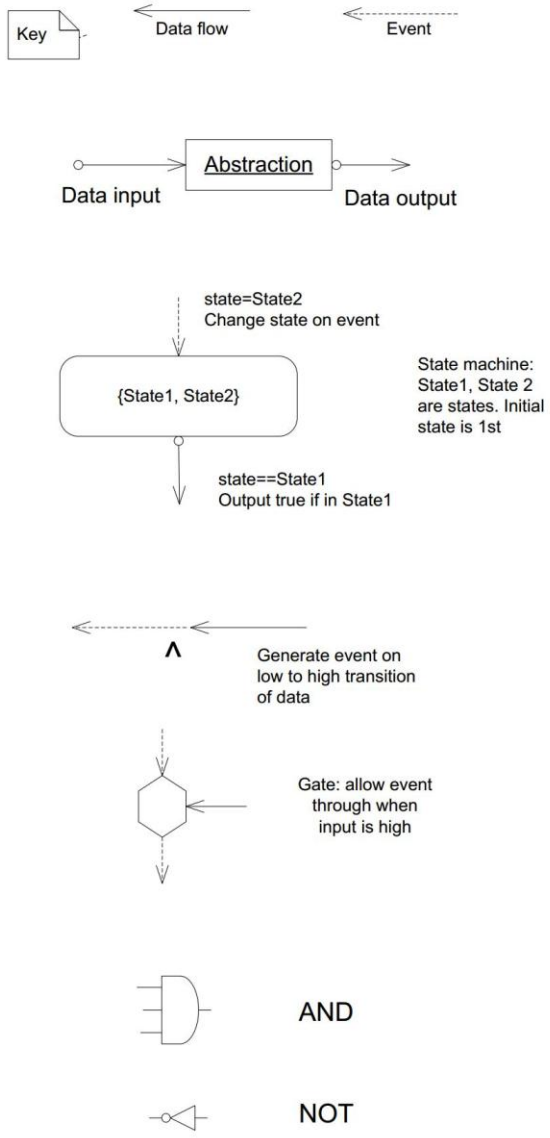
It is NOT an abstraction.
c.f. With the 'WarmerPlate' which has PotOnPlate and PotEmpty

Encapsulation by itself only works at compile time.
For read time, you must have abstraction.

CoffeeMaker.pdf


Controls its own heater wrt empty
UI
(Start button and Brewing light)
WaterBoiler
LampOn
ButtonPush (event)
On
Empty
OpenSteamReleaseValve
PotOnPlate
PotEmpty
WarmerPlate
Controls its own heater wrt PotOnPlate


UI
(Start button and Brewing light)
WaterBoiler
Controls own
Heating element
LampOn
ButtonPush (Event)
On
Empty
OpenSteamReleaseValve
Gate
^
state=Brewing
state==Brewing
{Idle, Brewing, Brewed}
state==Brewed
state=Brewed
state=idle
^
PotEmpty
PotOnPlate
WarmerPlate
Controls own
Warmer plate

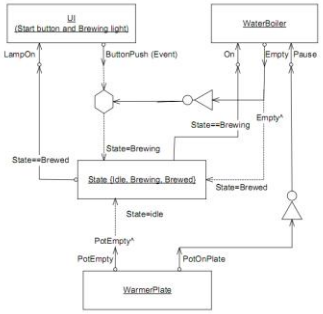

Key
Data flow
Event
Abstraction
Data input
Data output
state=State2
Change state on event
{State1, State2}
State machine:
State1, State 2
are states. Initial
state is 1st
state==State1
Output true if in State1
^
Generate event on
low to high transition
of data
Gate: allow event
through when
input is high
AND
NOT

```
// Knowledge dependencies :
// “Data flow computational model.doc”
#include “UserInterface.h”
#include “Boiler.h”
#include “WarmerPlate.h”

static enum {Idle, Brewing, Brewed} state;

void Poll()
{
        static bool prevBoilerEmpty, prevPotEmpty;

        if (UserInterface.Button && !Boiler.Empty && WarmerPlate.PotOnPlate && WarmerPlate.PotEmpty) {
                state = Brewing;
        }
        UserInterface.Button=false;
        Boiler.On = state==Brewing;
        if (Boiler.Empty && !prevBoilerEmpty) state = Brewed; prevBoilerEmpty = Boiler.Empty;
        UserInterface.LampOn = state==Brewed;
        Boiler.Pause = !WarmerPlate.PotOnPlate;
        if (WarmerPlate.PotEmpty && !prevPotEmpty) state = Idle; prevPotEmpty = WarmerPlate.PotEmpty;
}
```

Specialization

Application Layer

Coffee machine model 5000

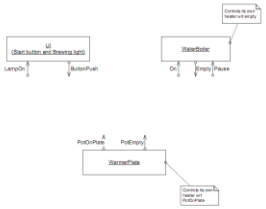


Knowledge Dependencies

API

Business abstraction layer

UI, Boiler, WarmerPlate



Increasing Knowledge Stability

API

Architectural abstraction layer

Data flow, State machine (computational models)

API

Foundation Abstraction layer

Increasing reuse

Abstraction

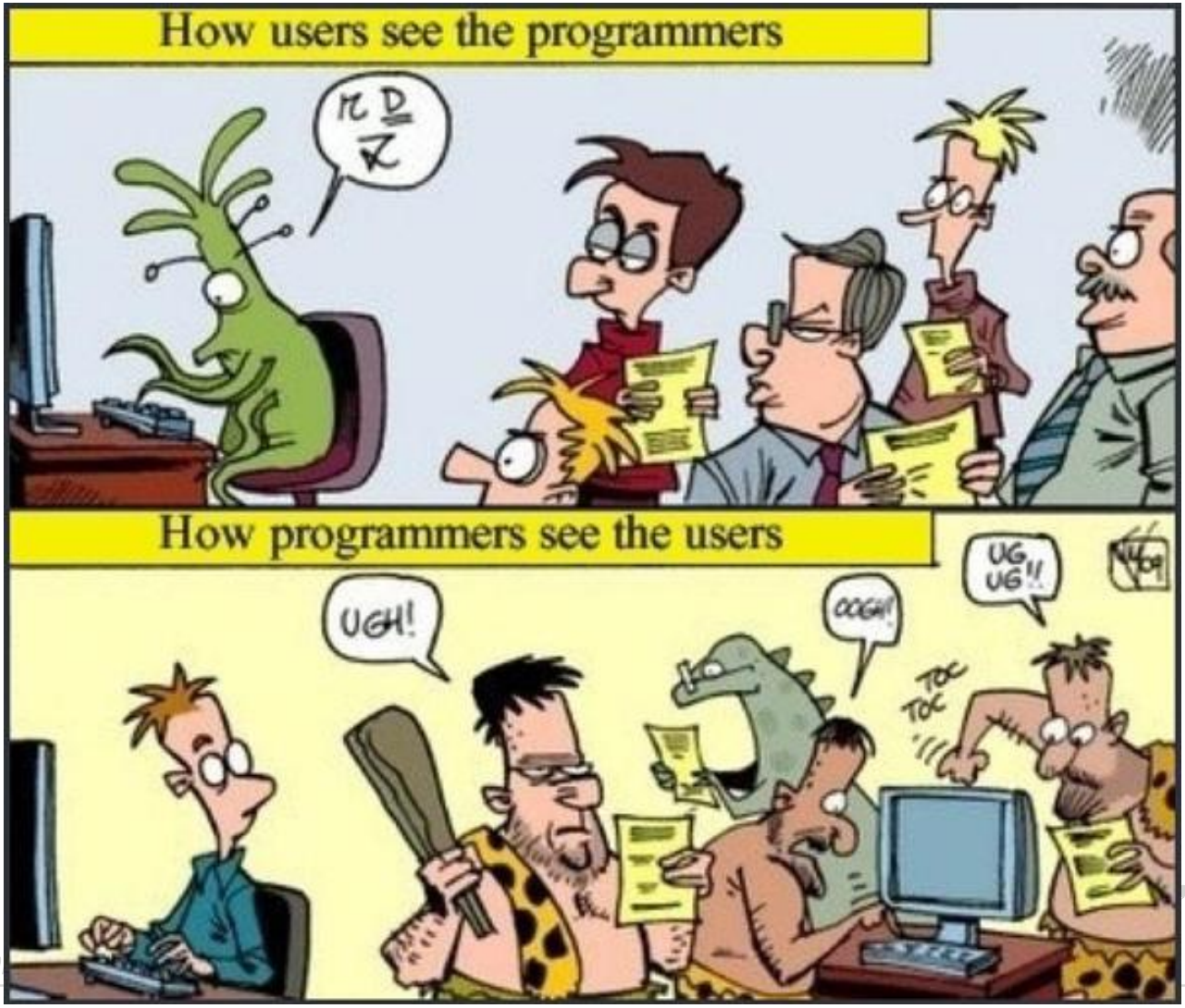
Architectural patterns

- Hidden wiring – Antipattern
- Invoke abstraction
- Owned Interfaces
- Shared Interfaces
  - Implicit wiring - Antipattern

Checklist

- Accidental complexity in top layer ~ 0
- The diagram is source code, not documentation
  - When you are modifying the app, you do it on the diagram
- Knowledge vs Runtime dependencies
- Make knowledge dependencies explicit
  - You cannot understand the code without the abstractions from the lower layers
  - Not only the 3 abstractions from the domain layer, but the abstraction of the architecture layer, the programming paradigms, including its syntax
  - Just as you would expect people to know the abstractions from the foundation layer e.g. C++
- Domain abstractions -> Solution space
  - The set of all possible applications I could write for this domain are supported by the domain abstractions.
  - This is what allows you to react quickly to changing requirements.
- Convention over Configuration
- Use diagrams for unstructured
  - into C or SystemC or Function Blocks, Labview, HDL
- Indirection always with abstraction
- Debugging through, reading through
- All files searches
- Mocking

Thank you


How users see the programmers
How programmers see the users
UGH!
OOGH!
UG UG!!
TOC TOC