

# Week 7

some additional material on scaffold

AI mark scheme

Tutorial announcements

## scaffold code is not a new idea

- Old timers had terrible debugging tools and needed to build small tests for subsystems
- Old timers knew to divide and conquer subsystems.
- Especially to find blunders causing regressions.
- Build stand alone tests that show fixes stay fixed.

# gnu regexp example

- <http://www.gnu.org/directory/regex.html>
- Used in many open source tools.
- Written by good people
- includes both static (regression) and interactive tests.

# Regex static test

from test/main.c

```
int main ()
{
    test_regress ();
    test_others ();
    test_posix_basic ();
    test_posix_extended ();
    test_posix_interface ();
    ...
}
```

# A Regexp Batch test

```
...from test/tregress.c
void test_regress ()
{
    ...
    /* enami@sys.ptg.sony.co.jp 10 Nov 92 15:19:02 JST */
    buf.translate = upcase;
    SIMPLE_MATCH ("A-[]", "A");
    buf.translate = NULL;

    /* meyerling@cs.utexas.edu Nov 6 22:34:41 1992 */
    simple_search ("\\w+", "a", 0);

    /* jimb@occs.cs.oberlin.edu 10 Sep 92 00:42:33 */
    buf.translate = upcase;
    SIMPLE_MATCH ("[\001-\0377]", "\001");
    SIMPLE_MATCH ("[\001-\0377]", "a");
    SIMPLE_MATCH ("[\001-\0377]", "\0377");
    buf.translate = NULL;
}
```

# regex interactive test

```
...from test/iregex.c
main ()
{
    ...
    printf ("String = ");
    gets (str);
    /* Now read the string to match against */
    scanstring (str);

    i = re_match (&buf, str, strlen (str), 0, &regs);
    printf ("Match value  %d.\t", i);
    if (i >= 0)
        print_regs (regs);
    putchar ('\n');

    i = re_search (&buf, str, strlen (str), 0, strlen (str), &regs);
    printf ("Search value %d.\t", i);
    if (i >= 0) print_regs (regs); putchar ('\n');
    ...
}
```

# regex test demo

- Not all the exciting to watch the static test
- The interactive test is primarily useful for debugging. (that's where I first met this code. In 1990 or so.)
- run `regex` and `iregex` in `regex-0.12/test`
- this was slide 6, in case I get lost in laptop..

# Java scaffold

- ClassPathUtil helps deal with CLASSPATH
- Bother to test in running JVM
- ClassPathUtil.main

```
public static void main(java.lang.String[] args) {  
    System.out.println("called with cwd " +  
                        new File(".").getAbsolutePath() );  
    System.out.println("Classpath = " + ClassPathUtil.getClasspath()  
    int i=0;  
    for( Enumeration e=ClassPathUtil.elements(); e.hasMoreElements();  
        System.out.println( "[" + ++i + "]" + (String)e.nextElement()  
    }  
}
```



# Test package

- Better to build a test class outside class to be tested
  - If no private methods need to be used..
  - Like in a package of test classes
- `util.SignalCatcher` is really inconvenient to test in running JVM.
- So create `util.test.SignalCatcherTest`

# Java test package

```
package org.zaleski.util.test;
import org.zaleski.util.SignalCatcher;
public class SignalCatcherTest {
    public static void main(String args[]){
        System.loadLibrary("SignalCatcher"); //force this to happen.
        SignalCatcher sm = new SignalCatcher();
        //arm the signal
        sm.addCatcherToCallback(2 /*SIGINT*/);
        //keep busy..while we fumble about doing a kill -HUP pid
        //in another window
        for(int i=0; i<20000000000; i++){
            for(int j=0; j<500000000; j++){
                //keep vm busy to simulate Jootch doing its thing
            }
            System.out.println("i=" +i);
        }
    } //main
}
```

# Interactive tests

- ivtools derived from Interviews
- C++ toolkit that was once hoped to become a standard in form of XC++
- Originally designed as part of a research program into GUI frameworks.
- lexi will be an important example.
- Meanwhile, here's the idemo toy program.

# idemo

- build/ivtools-1.0/src/idemo/DARWIN
- That's not broken. Those are overlays!
  - i.e. showing off.
- but it shows how a simple program can be put together to test drive widgets..

# idemo

```
class App {
public:
    App();      ~App();
    int run(int, char**);
    void open();    void save();    //..

private:
    WidgetKit* kit_;
    LayoutKit* layout_; //..
    ApplicationWindow* main_;
    FileChooser* dialog_;
    Menu* menubar();
    Menu* make_menu(Menu*, CommandInfo*, int = 0);
    MenuItem* make_submenu(MenuItem*, Menu*);
    MenuItem* make_item(MenuItem*, Action*);
    void add(const char* label, Glyph*);
};
```

# Component architecture evident in tests..

just a sample of the general style of iv

```
TelltaleGroup* group = new TelltaleGroup;
add(
    "Radio buttons",
    layout.vbox(
        kit.radio_button(group, "Able", action),
        vspace4,
        kit->radio_button(group, "Baker", nil),
        vspace4,
        kit->radio_button(group, "Charlie", nil)
    )
);
```

running idemo we can see these on the left

# idraw

- idraw was a totally incredible program in its time.
- Remarkably little fuss about IV??
- Can't resist a little demo..
- This is a lot like the upcoming lexi case study.

# AI

- Lesson was supposed to have been that small inconsistencies are hard to avoid and relatively easy to work around
- It turned out that test scaffold was real lesson.
- In isolated, academic environment, inconsistencies bothered you a lot.



# AI marks

Number of marks: 91

Minimum, Maximum, Range: 0, 40, 40

Mean: 30.9264

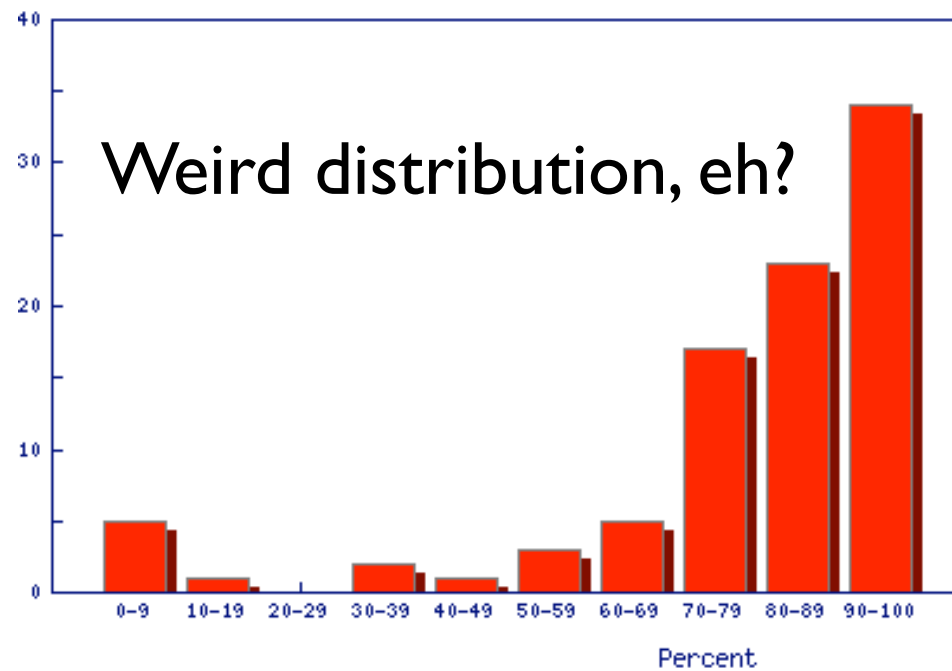
Std. Dev.: 9.79533

Quartiles: 29.00, 34.00, 37.00

## Mark Breakdown

Mark	Number	Percent
-----		
A	57	63%
B	17	19%
C	5	5%
D	3	3%
E & F	9	10%
-----		

Mark Distribution



# Analysis vs Design

- AI was about implementation
- Rosenberg et al did
  - Analysis
    - Use cases and Domain model
  - Preliminary Design
    - Robustness
  - Design
    - Sequence
    - Static model
  - Many people confuse analysis and Design

# AI mark scheme

marks	
10	Understanding the entity design. No flagrant departures from design. if documented properly you may accept variations. The key is that they understood the design not that they followed it slavishly.
10	testing covers all use cases and exceptional courses. You should be able to discern this entirely from the document. If you have to fish around in the code you can penalize .
10	Quality of test scaffold design. If anyone thought to create a UML for their scaffold design reward it! Test scaffold must not disturb the entity class implementation whatsoever. Put another way, the testing must be done from outside the entity classes. I characterized this by telling them that their entity class implementations are to be “handed off” to the user interface team later in this hypothetical project.
10	Implementation quality. This is where the keeners who made great interactive programs or test scaffold should do a little better. If students did a really good job of documenting their implementation in a way that referred to the design you can reward them here also. Particularly messy implementations can be penalized.

# Approaches to AI scaffold

- Driver Classes
  - Like the Java examples presented above
- Interactive Boundary classes
  - Hard! A lot of work!
  - How effective relative to driver classes?

# AI comments?

- What did you learn in AI?
- What should we tell Rosenberg?
- How should we have set up the assignment?

## A2

- Confusion exists about what is analysis
  - Not our fault. Confusion exists!
- Analysis is about the domain
  - NOT about the proposed software.
- Use cases, Domain classes
  - NO robustness or sequence diagrams.

# New fangled tractors

- John Deere 9560.
- > 300Hp
- Base price \$usd 158K
- Greenstar ~\$5000



## A2 is science fiction

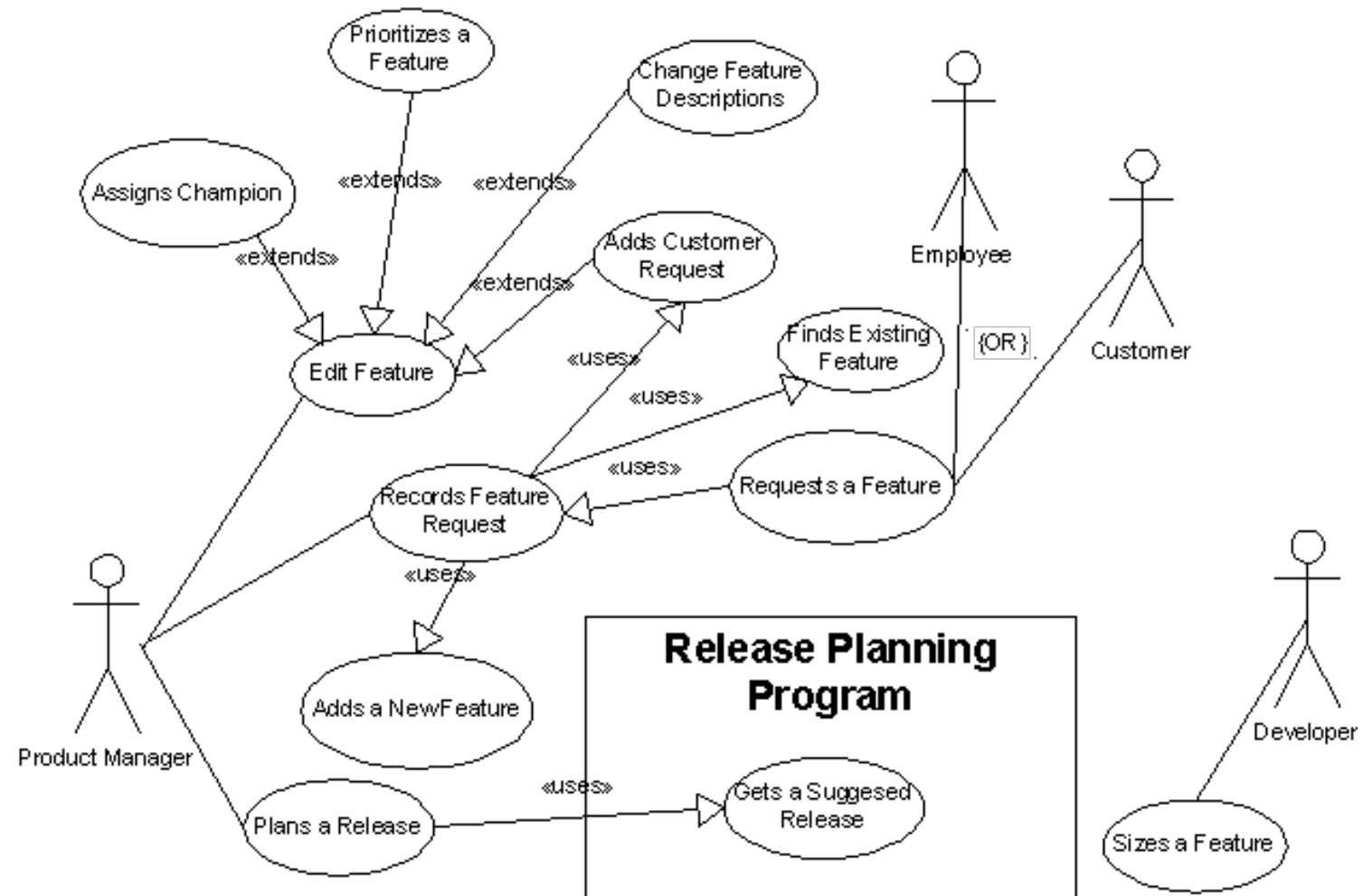
- It's the software.
- Idea is that if every “quad” of field is monitored yield can be improved relative to fixed costs of equipment and energy.
- You are primarily analyzing the historical data.
- “agricultural analytics” makes fun of financial derivatives software which is referred to as “analytics”.



## Intended A2 lessons.

- Getting language in use cases in sync with domain model surprisingly difficult.
- Controlling scope of analysis challenging.
  - If you find yourself modeling too much think about scope.
- Remember planaria use case diagram?
  - Only required to support a few use cases.

# use case scope..



## A2

- (Overview) Crop plan has always been done
- Detailed crop plan feeds the implements.
- “Analytics” totally out of scope.
- Do not design relational database!
  - Describe historical data in terms of classes.
  - DB analysts will figure out how to “persist” your model later, during design.