SOFTENG 701:
Advanced Software Engineering Development Methods

Lecture 1.01: Evidence for Design Advice

Ewan Tempero
Department of Computer Science

Agenda

- Admin
  - ?
- Intro to part 1
- Evidence for design advice

Who is Ewan Tempero?

- Associate Professor @ UoA since 2002
- In Computer Science department, but teaching almost entirely in SOFTENG
- Past life: University of Otago (BSc Hons), University of Washington (PhD), Victoria University of Wellington (Lecturer)
- Research interests: measuring design quality
- Other: Software Engineering as a Profession

Assessment Part 1

- Assignment 1, Due Monday 18th March, worth 10% — Development and report on modifiable designs for Mancala
- Assignment 2, Due Friday 19th April Worth 20% — Evidence for design advice
- Test 1, Wednesday 17th April, Worth 20%

Assignment 1: Mancala

- Played with 48 seeds on a board with 2 stores and 12 houses.
- Each player controls 1 store and 6 houses
- A game starts with 4 seeds in each house
- Game play
  - A turn consists of 1 or more moves
  - A move consists of taking seeds from a house and planting 1 in each house anti-clockwise, including the player's store (but not opponent's)
  - If a move ends in the player's store, the player gets another move
  - If a move ends in player's house that was empty, the player gets that seed and all seeds in the opposing house
  - The game ends when one player only has empty houses
  - The score is all the seeds in a player's store and all houses (highest score wins)

Mancala Example

```
+----+-------+-------+-------+-------+-------+-------+----+
| P2 | 6[ 4] | 5[ 4] | 4[ 4] | 3[ 4] | 2[ 4] | 1[ 4] |  0 |
|    |-------+-------+-------+-------+-------+-------|    |
|  0 | 1[ 4] | 2[ 4] | 3[ 4] | 4[ 4] | 5[ 4] | 6[ 4] | P1 |
+----+-------+-------+-------+-------+-------+-------+----+
Player 1's turn - Specify house number or 'q' to quit: 3
+----+-------+-------+-------+-------+-------+-------+----+
| P2 | 6[ 4] | 5[ 4] | 4[ 4] | 3[ 4] | 2[ 4] | 1[ 4] |  1 |
|    |-------+-------+-------+-------+-------+-------|    |
|  0 | 1[ 4] | 2[ 4] | 3[ 0] | 4[ 5] | 5[ 5] | 6[ 5] | P1 |
+----+-------+-------+-------+-------+-------+-------+----+
Player 1's turn - Specify house number or 'q' to quit: 5
+----+-------+-------+-------+-------+-------+-------+----+
| P2 | 6[ 4] | 5[ 4] | 4[ 4] | 3[ 5] | 2[ 5] | 1[ 5] |  2 |
|    |-------+-------+-------+-------+-------+-------|    |
|  0 | 1[ 4] | 2[ 4] | 3[ 0] | 4[ 5] | 5[ 0] | 6[ 6] | P1 |
+----+-------+-------+-------+-------+-------+-------+----+
Player 2's turn -  Random chooses house #6
+----+-------+-------+-------+-------+-------+-------+----+
| P2 | 6[ 0] | 5[ 4] | 4[ 4] | 3[ 5] | 2[ 5] | 1[ 5] |  2 |
|    |-------+-------+-------+-------+-------+-------|    |
|  1 | 1[ 5] | 2[ 5] | 3[ 1] | 4[ 5] | 5[ 0] | 6[ 6] | P1 |
+----+-------+-------+-------+-------+-------+-------+----+
```

Mancala Example

```
Player 1's turn - Specify house number or 'q' to quit: 6
...
Player 2's turn -  Random chooses house #1
...
Player 2's turn -  Random chooses house #5
+----+-------+-------+-------+-------+-------+-------+----+
| P2 | 6[ 2] | 5[ 0] | 4[ 6] | 3[ 7] | 2[ 7] | 1[ 0] |  3 |
|    |-------+-------+-------+-------+-------+-------|    |
|  3 | 1[ 6] | 2[ 6] | 3[ 2] | 4[ 6] | 5[ 0] | 6[ 0] | P1 |
+----+-------+-------+-------+-------+-------+-------+----+
Player 1's turn - Specify house number or 'q' to quit: 3
+----+-------+-------+-------+-------+-------+-------+----+
| P2 | 6[ 2] | 5[ 0] | 4[ 6] | 3[ 7] | 2[ 0] | 1[ 0] | 11 |
|    |-------+-------+-------+-------+-------+-------|    |
|  3 | 1[ 6] | 2[ 6] | 3[ 0] | 4[ 7] | 5[ 0] | 6[ 0] | P1 |
+----+-------+-------+-------+-------+-------+-------+----+
```

Test Infrastructure

```
public class Mancala {
  public static void main(String[] args) {
    new Mancala().play(new MockIO());
  }
  public void play(MockIO io) {
    io.println("+----+-------+-------+-------+-------+...");
    io.println("| P2 | 6[ 4] | 5[ 4] | 4[ 4] | 3[ 4] |...");
    io.println("|    |-------+-------+-------+-------+...");
    io.println("|  0 | 1[ 4] | 2[ 4] | 3[ 4] | 4[ 4] |...");
    io.println("+----+-------+-------+-------+-------+...");
    io.println("Player 1's turn - Specify house number or 'q'...");
  }
}
```

- `MockIO` provides input from keyboard and output to console

Test Infrastructure

```
...
io.setExpected("test/simple_start.txt");
io.println("+----+-------+-------+-------+-------+...");
...
```

- `MockIO` can compare I/O actually done with expected I/O specified by a file
- E.g., in `test/simple_start.txt`:

```
# Do one move, which doesn't affect the player's store, then quit.
>+----+-------+-------+-------+-------+-------+-------+----+
>| P2 | 6[ 4] | 5[ 4] | 4[ 4] | 3[ 4] | 2[ 4] | 1[ 4] |  0 |
>|    |-------+-------+-------+-------+-------+-------|    |
>|  0 | 1[ 4] | 2[ 4] | 3[ 4] | 4[ 4] | 5[ 4] | 6[ 4] | P1 |
>+----+-------+-------+-------+-------+-------+-------+----+
>Player 1's turn - Specify house number or 'q' to quit:
<1
...
```

Deliverables

- Single archive file (zip or tar only — NOT zipx, rar, etc!) named with your UPI.
- Implementation of Mancala that passes provided test cases
  - source code organised in packages (can be easily compiled)
  - runnable jar file.
- Report (PDF) describing your design and justifying why it supports modifiability
  - include pictorial description (simplified UML)
  - description of purpose of each class (or interface)
  - explanation of what you mean by modifiability
  - formatted according to IEEE template
- Submission procedure to be confirmed.
- More details on Cecil

Design Advice

- Big classes are bad

Design Advice

- Big classes are bad
- Why should we believe this?
  - Wikipedia says so `http://en.wikipedia.org/wiki/Code_smell`

Design Advice

- Big classes are bad
- Why should we believe this?
  - ~~Wikipedia says so `http://en.wikipedia.org/wiki/Code_smell`~~

Design Advice

- Big classes are bad
- Why should we believe this?
  - ~~Wikipedia says so `http://en.wikipedia.org/wiki/Code_smell`~~
  - Some famous person said so: **Martin Fowler** “Refactoring: Improving the Design of Existing Code” Addison-Wesley 1999. p78

Design Advice

- Big classes are bad
- Why should we believe this?
  - ~~Wikipedia says so `http://en.wikipedia.org/wiki/Code_smell`~~
  - Some famous person said so: **Martin Fowler** “Refactoring: Improving the Design of Existing Code” Addison-Wesley 1999. p78
- Is it enough that some famous person said so?
  - Maybe it was true when the person said so, but not now
  - Maybe it was true for only some situations (e.g., when there is no good IDE)
  - Maybe that person is just plain wrong. . .

  ⇒ need good evidence to support such claims

Design Advice

- Big classes are bad
- Why should we believe this?
  - ~~Wikipedia says so `http://en.wikipedia.org/wiki/Code_smell`~~
  - Some famous person said so: **Martin Fowler** “Refactoring: Improving the Design of Existing Code” Addison-Wesley 1999. p78
- Is it enough that some famous person said so?
  - Maybe it was true when the person said so, but not now
  - Maybe it was true for only some situations (e.g., when there is no good IDE)
  - Maybe that person is just plain wrong. . .

  $\Rightarrow$ need good evidence to support such claims
- Is there good evidence to support such claims? Two answers:
  - “Yes”
  - No!

What is good evidence?

- Historical — from observations based on the past, we see that it is often the case that bad software (designs) contain big classes
  - How good are the observations?
  - Do the conditions that existed when the past observations were made still hold today?
  - How often is it the case — 100%? 95%? 50%? If it is only 30% does that invalidate the claim?
  - Is the presence of big classes the cause or the effect of bad software?
  - prediction = what was true in the past is still true now

What is good evidence?

- Historical — from observations based on the past, we see that it is often the case that bad software (designs) contain big classes
  - How good are the observations?
  - Do the conditions that existed when the past observations were made still hold today?
  - How often is it the case — 100%? 95%? 50%? If it is only 30% does that invalidate the claim?
  - Is the presence of big classes the cause or the effect of bad software?
  - prediction = what was true in the past is still true now
- Explanatory — we have a explanation (“model”) for why big classes leads to bad software
  - How do we know the explanation is correct?
  - How do we know the explanation applies to all software (or at least our software)?
  - The model description tells us which is cause and which is effect
  - prediction = the model tells us what will happen

Questions not answered

- What is “big”?
- What is “bad”?

State of Art

- Explanatory
  - often no models exist
  - vague on the exact mechanism relating cause (advice to follow) and effect (impact on design quality)
  - models too simple to be useful
  - almost no evidence that models are “valid”
- Historical
  - most common form
  - often not formal (“expert advice”)
  - often no scientific evidence in support
  - what evidence there is is weak or inconsistent

Course (Part 1) Goals

- Investigate what good design might mean, and improve our understanding of what it should mean
  - Learn a little bit of history of thinking on good design — where did some design ideas come from and are they still valid
  - Learn some new ideas about how to do good design — what other cool ways are there to do designs
  - Learn a little bit on how to evaluate how good a design is — we have a design, but how do we know it is any good
  - See a few bad designs - how to recognise known bad designs.
- Determine what (good) evidence there is, that is, following the advice leads to better designs.
  - How is design quality being measured?
  - How do we evaluate the evidence that exists regarding the design advice?
  - What evidence can we add to confirm (or refute) the claims regarding the design advice?

Intermission

A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: “What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise.” The scientist gave a superior smile before replying, “What is the tortoise standing on?” “You’re very clever, young man, very clever,” said the old lady. “But it’s turtles all the way down!”

A Brief History of Time, Stephen Hawking, 1998

Science in a Nutshell

- Create a model that explains reality
- Use the model to predict the future.

Science in a Nutshell

- Create a model that explains reality
- Use the model to predict the future.
- Questions:
  - Where does the model come from?
  - How do we know how good the model is?

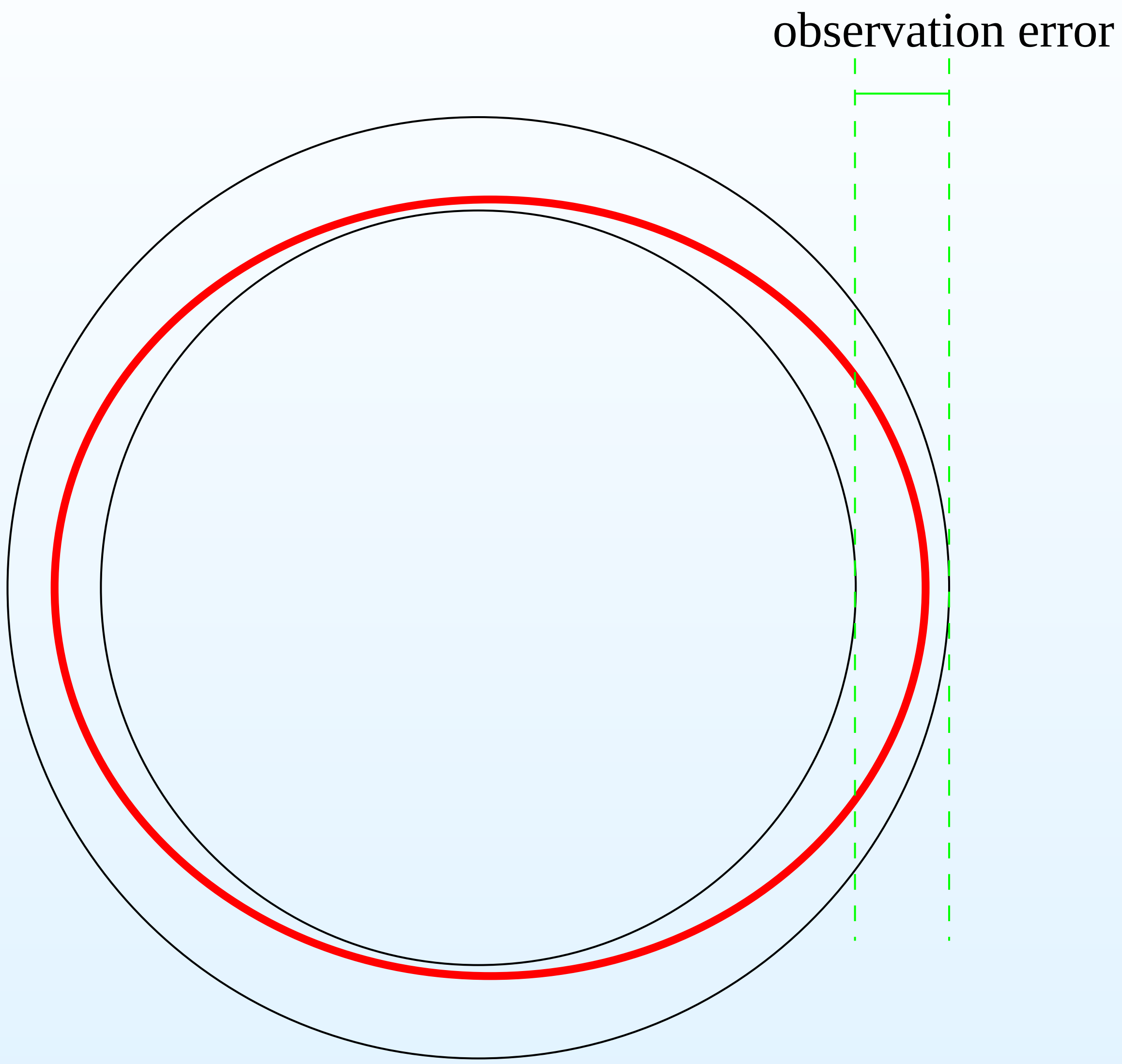
Science in a Nutshell

- Create a model that explains reality
- Use the model to predict the future.
- Questions:
  - Where does the model come from?
    $\Rightarrow$ observations
  - How do we know how good the model is?
    $\Rightarrow$ observations

Modelling the Cosmos

- Turtles — What predictions can we make? How can we falsify this model?
- Historical — *"It has been 364 days since the sun rose with its left edge aligned to the rock that looks like a gazelle. Based on what has happened every 365 days previous to that, I predict the sun will rise tomorrow aligned in the same way"*. Doesn't explain comets
- Geocentric model — *The heavenly bodies are on concentric rotating celestial spheres with the (stationary) Earth in the centre.* Doesn't explain retrograde motion
- Copernican heliocentrism — *Celestial spheres but with the Sun at the centre* Prediction of locations still not exact
- Kepler's laws of planetary motion — *Not spheres, but ellipses, and a description of how fast planets move* Much better prediction, but still occasionally incorrect, and those pesky comets.
  Describes how planets move, but does not **explain**
- Newton's laws of motion — *Gravity, inverse law* Even better prediction, comets "explained", but still occasional errors (position of Mercury)
  Orbital motion is the **effect**, gravity is the **cause**
- Einstein's general theory of relativity — *Frames of reference, gravity as a geometric property of space and time*

Observations



(Explanatory) Model Building

- Look at the world (make observations)
- Come up with an explanation for what we see (build the model)
- Make a prediction about what will happen in the future based on the model
- Observe the future. If it differs from the prediction by more than the measurement error and model error, the model is inadequate.

But! If the predictions match, this does not mean the model is “correct”

$\Rightarrow$

- Need to be able to make observations $\Rightarrow$ measurement
- Need to be able compare measurements sensibly
- Need to know measurement error
- Need to be able to understand what other explanations are for the observations made (*threats to validity*)