PROGRAMMING PATTERNS AND PROCESS

CITS1001

Motivation

- “The labs were all really interesting and well paced. Large seemingly impossible tasks were broken down into manageable chunks.”
  - CITS1001 student 2012

- Question addressed in this lecture: how to break “large, seemingly impossible tasks” into manageable chunks.

Scope of this lecture

- What are programming patterns?
- Why use patterns ?
- Some useful elementary patterns:
  - Whether-or-not
  - Process-all-items
  - Variable use patterns

- Program development processes:
  - Waterfall, Agile, STREAM

PROGRAMMING PATTERNS

Compare and Contrast

```
public void depositA(int amount)
{
   if ( amount > 0 ) {
      balance = balance + amount;

      if (balance > maxBalance) {
            maxBalance = balance;
       }
   }
}
```

```
public void depositB(int amount)
{
   if (amount < 0) {
            balance = balance;
   } else {
         balance += balance;
   }

  if (maxBalance < balance) {
            maxBalance = balance;
  }
}
```

Programming Patterns

- Programming patterns correspond to fragments of code that accomplish common programming goals
- Recognising and applying patterns is one of skills that distinguishes expert programmers from novices

Whether Or Not Selection

- Bergen describes the Whether Or Not pattern as follows:

  “You are in a situation in which some action may be appropriate or inappropriate depending on some testable condition. ... You don't need to repeat the action, only to decide Whether Or Not it should be done. There are no other actions to do instead of this one. You want to write simply understood code.”

- For example,

```
if (amount > balance) {
   balance += amount ;
}
```

Whether Or Not Selection (cont)

- Barnes and Kölling use a variant of the Whether Or Not pattern in which an else branch containing only print statement(s) is used to inform the user of the error condition. For example,

```
if (amount > balance) {
   balance += amount ;
} else {
   System.out.println("amount must be >0");
}
```

Process All Items

- Astrachan and Wallingford describe the Process All Items pattern as: “The items are stored in an array a. Use a definite loop to process all the items.”.
- For example,

```
for (int k=0; k < a.length; k++) {
 a[k] = a[k]*2;
}
```

- Barnes and Kölling use a for-each version of this pattern for collections (and arrays since Java 5)

```
for (Atype ak : a) { process ak; }
```

Temporary Variables

- Kent Beck explains “Temporary variables let you store and reuse the value of expressions. They can be used to improve the performance or readability of methods.”

- See http://c2.com/ppr/temps.html for
  - Temporary variable
  - Collecting temporary variable
  - Caching temporary variable
  - Explaining temporary variable
  - Reusing temporary variable
  - Role suggesting temporary variable name

Temporary Variable Pattern Examples

- Collecting

```
int sum = 0;
for (int ai : a) { sum = sum + ai; }
```

- Caching

```
int avg = average(a); int var = 0;
for (int ai : a) {
   var = var + Math.Pow(ai-avg, 2);
}
```

- Explaining

```
if (a[j] < min) {
     min = a[j];
     swappos = j;
}
```

Further Reading

- **Loop patterns: Definite process all items**, O. Astrachan and E. Wallingford. 1998. Retrieved March 2012 from http:// www.cs.duke.edu/ ola/patterns/plopd/loops.html
- **Portland pattern repository: Temporary variables**, K. Beck. 1995. Retrieved March 2012 from http://c2.com/ppr/temps.html
- **Patterns for Selection** Version 4, J. Bergin., 1999. Retrieved March 2009 from http://csis.pace.edu/ bergin/patterns/ Patternsv4.html

- **Design Patterns: Elements of Reusable Object-Oriented Software**, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, (the "Gang of Four"), Addison Wesley, 2004

SOFTWARE DEVELOPMENT PROCESS

Outline

- Software Development Processes
  - top down; bottom up; islands of functionality
- Stepwise improvement activities
  - restructure; refine; extend
- STREAM: a systematic process for novices

- Reference:
  STREAM: A First Programming Process,
  M.E. CASPERSEN and M. KOLLING,
  ACM Transactions on Computing Education, 9(1), March 2009
- http://dl.acm.org/citation.cfm?id=1513597

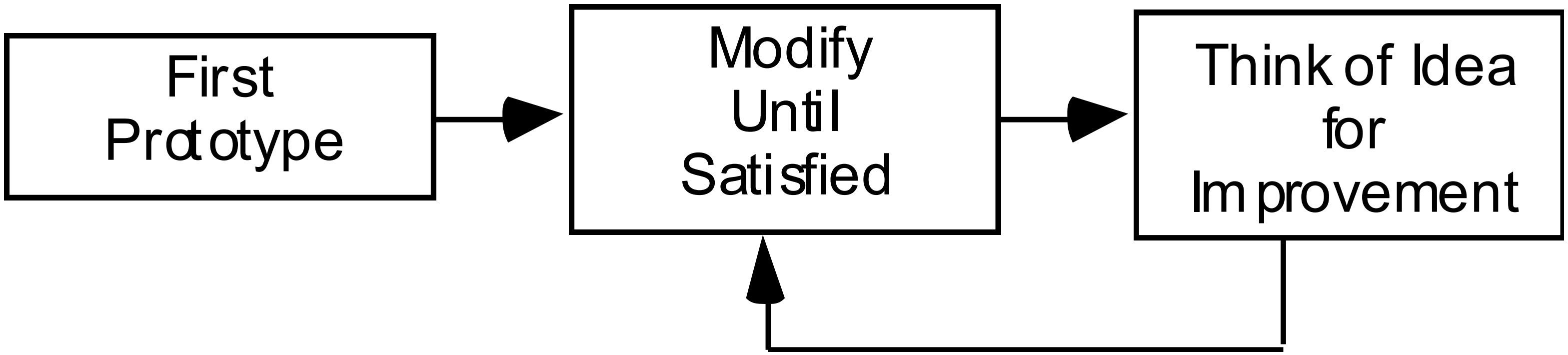
Some important terms

- Process (n)
  - a series of actions or operations designed to achieve an end
- Product (n)
  - something produced by a natural or artificial process
  - a saleable or marketable commodity
- Lifecycle (n)
  - the series of stages in form and functional activity through which an organism passes during its lifetime

Software Development Lifecycle (SDLC)

- A model for the process of developing SW
- Three standard models (you should know)
  - opportunistic (aka chaotic)
  - waterfall
  - iterative (agile)
- All SDLCs involve steps for requirements, design, implementation and quality assurance

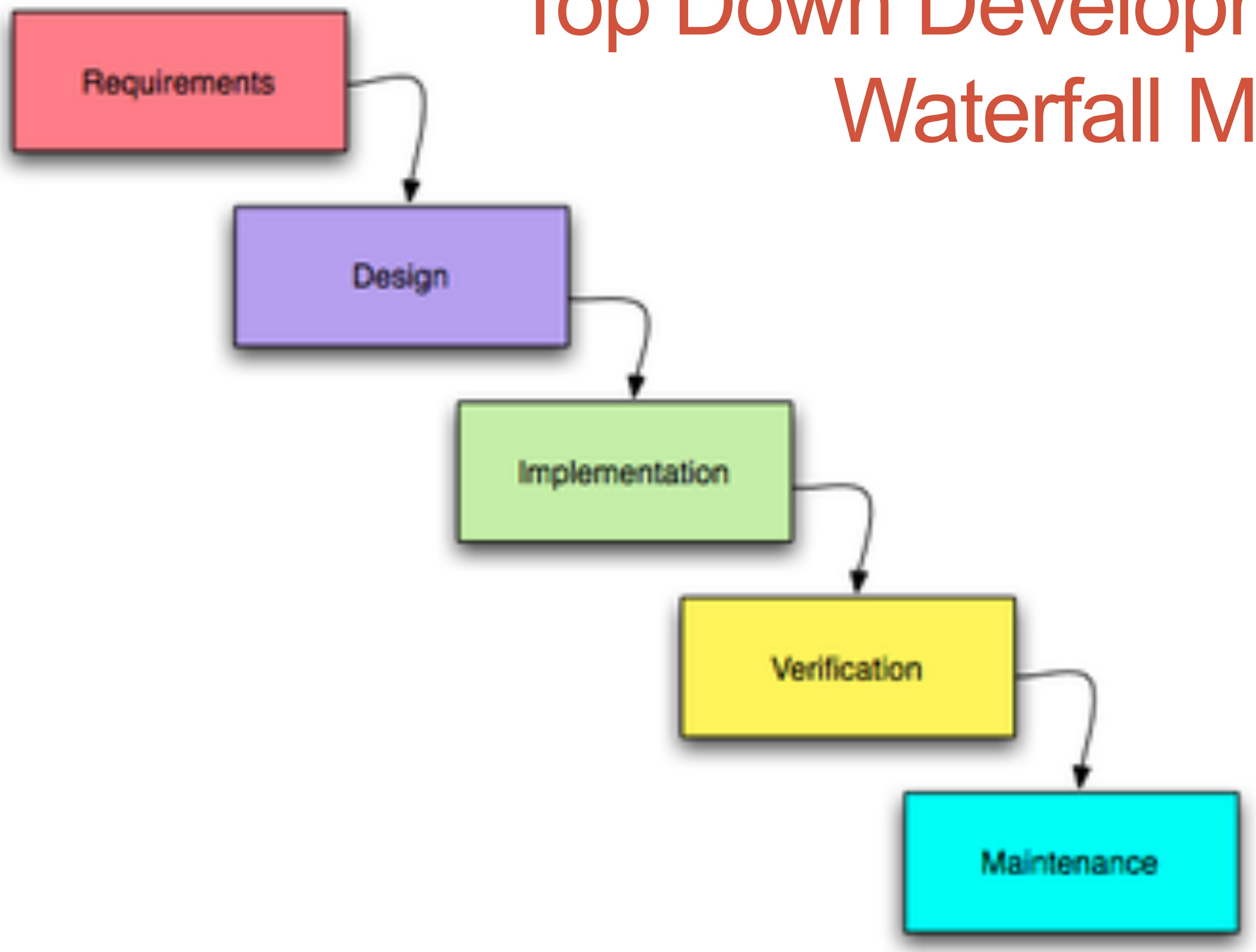
Chaotic (opportunistic) SDLC



Chaotic approach (not recommended)

- Does not acknowledge the importance of working out the requirements and the design before implementing a system
- Since there are no plans, there is nothing to aim towards, or measure against
- No explicit recognition of the need for systematic testing or other forms of quality assurance.
- Leads to … very high cost of developing and maintaining software with the opportunistic approach

Top Down Development
Waterfall Model



Waterfall model limitations

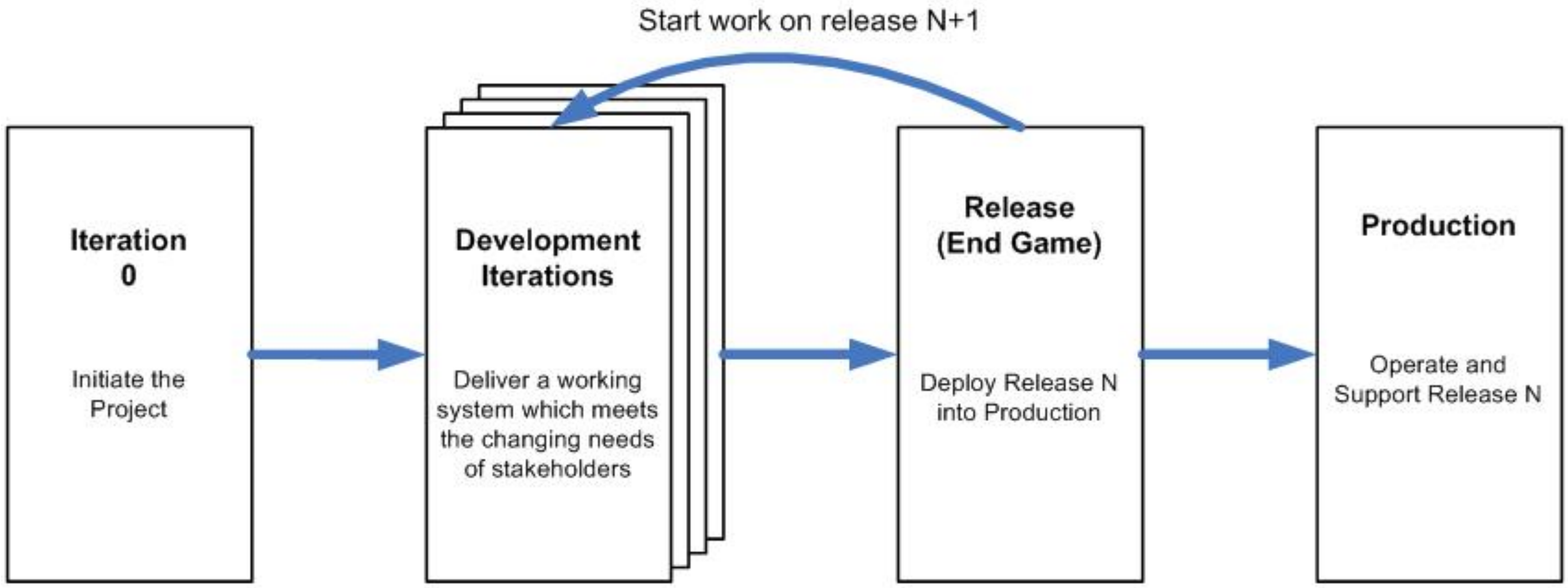
- The model implies that you should attempt to complete a given stage before moving on to the next stage
  - Does not account for the fact that requirements constantly change.
  - It also means that customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- The model implies that once the product is finished, everything else is maintenance.

Agile Development Techniques

- **Test Driven Development** – because each iteration must deliver working software it must work.
- **Short iterations** – from 2 weeks to 2 months.
- Strong **customer** involvement
- **Adaptable process** allowing for requirements to
- Change

- Agile and Waterfall are opposite ends of the "control" spectrum

Agile Development



- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements modeling
- Initial architecture modeling
- Setup environment

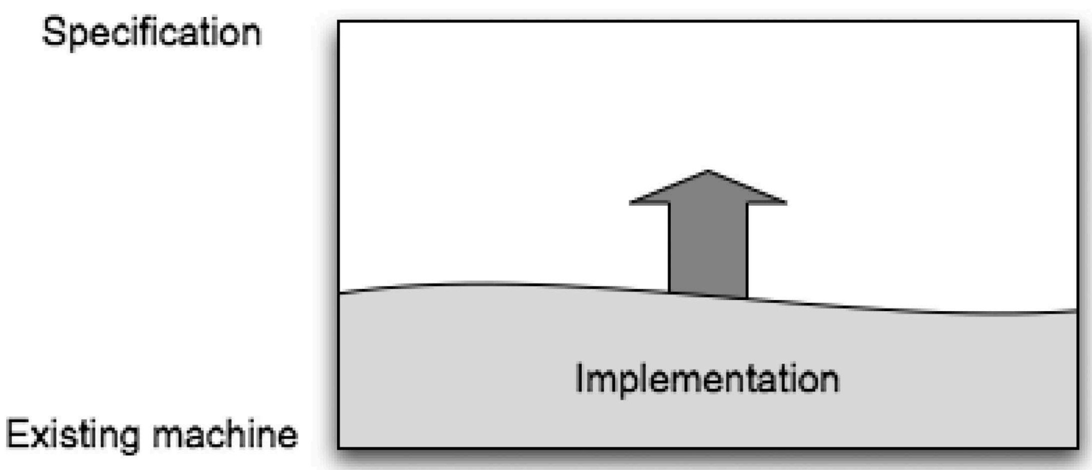
- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Confirmatory testing
- Investigative testing
- Evolve documentation
- Internally deploy software

- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production

- Operate system
- Support system
- Identify defects and enhancements

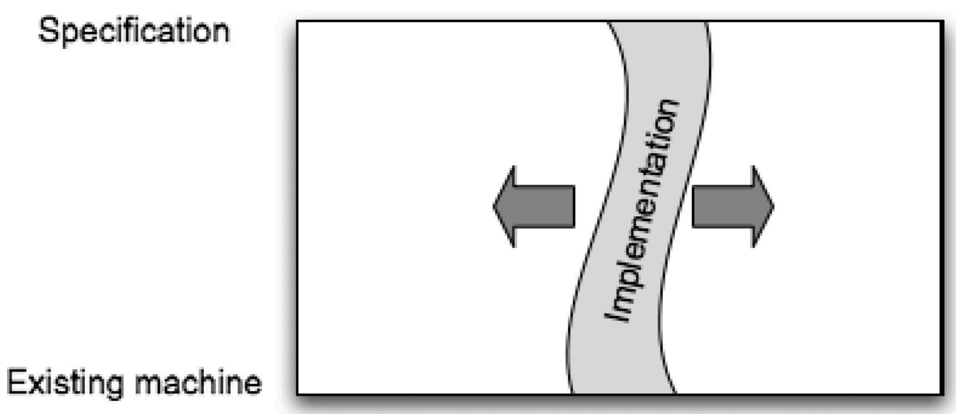
Bottom Up Development

- From 1990s OO programming
- Lower level components developed first
- Higher-level functionality slowly built on top of the low-level modules



Growing Islands of Functionality

- Initially implement small subsets of functionality completely (from the user interface down to the available machine)
- Function is gradually increased by growing the available islands of implementation

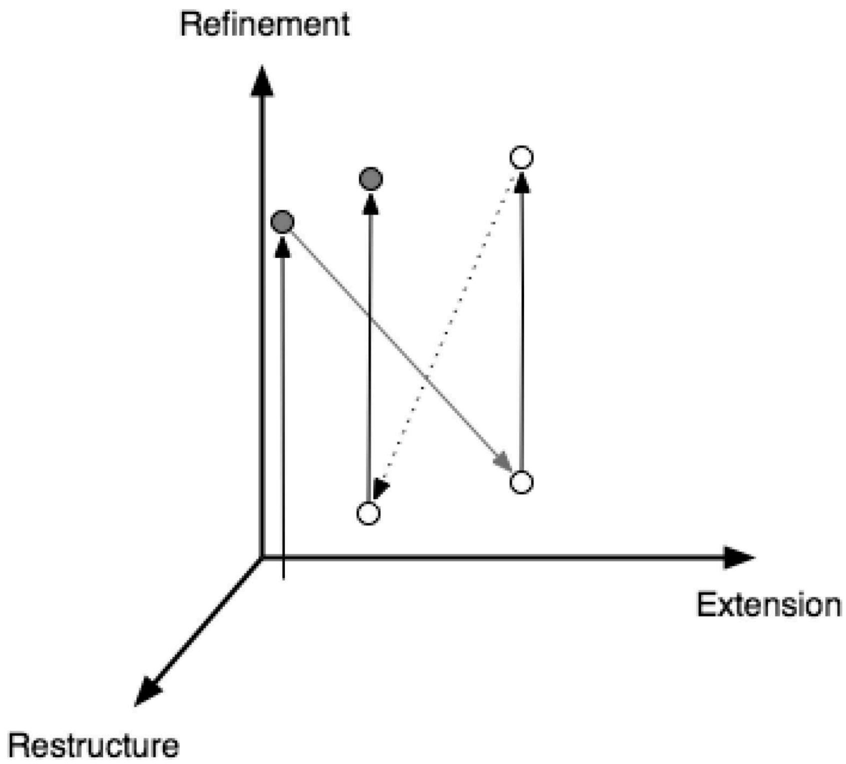


STEPWISE DEVELOPMENT

Stepwise Development Activities

- **Extend** the specification to cover more (use-) cases;
- **Refine** abstract code to executable code to meet the current specification;
- **Restructure** improves nonfunctional aspects of a solution without altering its observable behavior: design improvements through refactoring, efficiency optimisation, or portability improvements.

Stepwise Development



STREAM DEVELOPMENT PROCESS

Reference:
STREAM: A First Programming Process,
M.E. Caspersen and M. Kölling,
ACM Transactions on Computing Education, 9(1), March 2009
http://dl.acm.org/citation.cfm?id=1513597

STREAM summary (1)

- Step 1: Stubs
  - Create a Skeleton Class with Method Stubs
- Step 2: Tests
  - Ensure that Tests Are Available
- Step 3: Representations
  - Consider Alternative Representations
- Step 4: Evaluation
  - Evaluate the Alternative Representations

STREAM summary (2)

- Step 5: Attributes
  - Define Instance Fields
- Step 6: Methods
  - Implement the Methods

```
while there is an unfinished method:
   Pick an unfinished method;
   //Implement the method
   while not done:
          improve the method;
          test;
```