CS342: Software Design

August 29, 2017

Gang Wei

- Adjunct Lecture, Co-teaching with Prof. Patrick Troy
- Currently at HighGround Inc.
- Echo Global Logistics, Accenture
- Ph.D. 2001, Wayne State University
- 937 SEO, 1:55 PM ~ 4:00 PM
- gang.g.wei@gmail.com

By end of this course, you will have...

Deeper understanding of OOP and software design patterns

Choose the right pattern for your task!

Write clean code

Enhance potential in R & D career

Analyze software requirements, find solution, and take ownership

Learn teamwork and leadership

Class information

- Website: https://www.cs.uic.edu/bin/view/CS342/Fall2017
- Late assignments will be accepted with the following penalties:

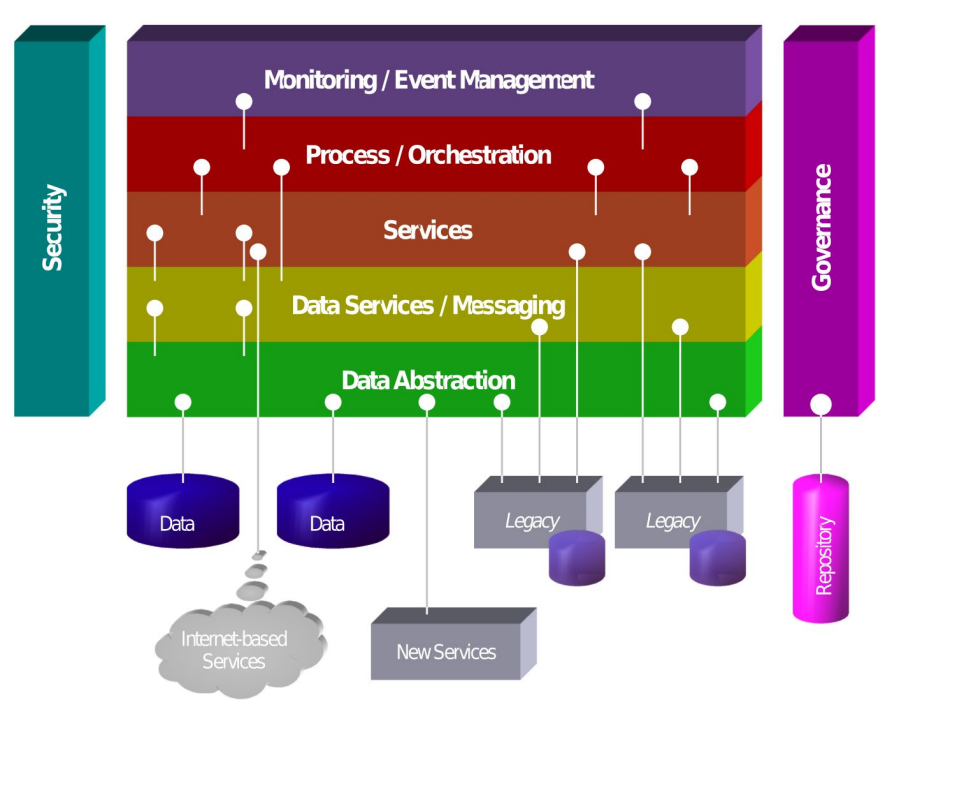
One Day Late:	10% penalty
Two Days Late:	30% penalty
Three Days Late:	60% penalty
Four+ Days Late:	100% penalty (i.e. a score of 0 is recorded)

- Programs that do not compile will receive a grade of 0
- We do MOSS!
- Programming teams

OOP & Design pattern goes a long way to your success

- Sets CS students apart from other students
- Backend developer & architecture require high skill levels in this
- Principles in OOP & Design pattern never gets outdated
- Software craftsmanship
- The earlier the better!

Service Oriented Architecture



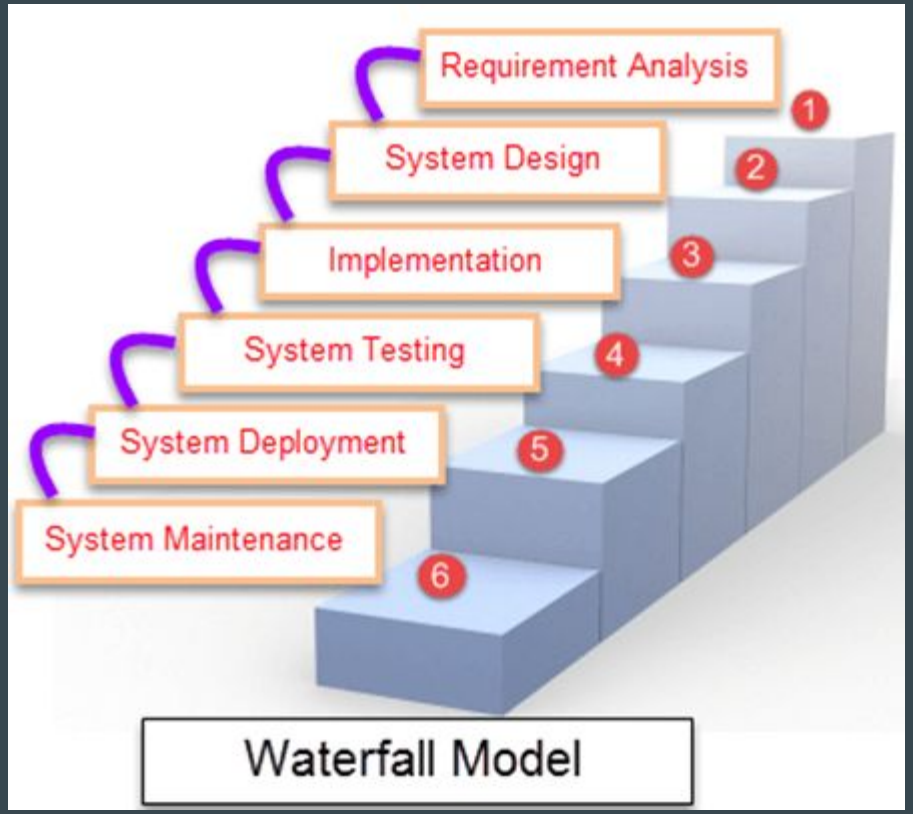
Service Layer

Business Logic Layer

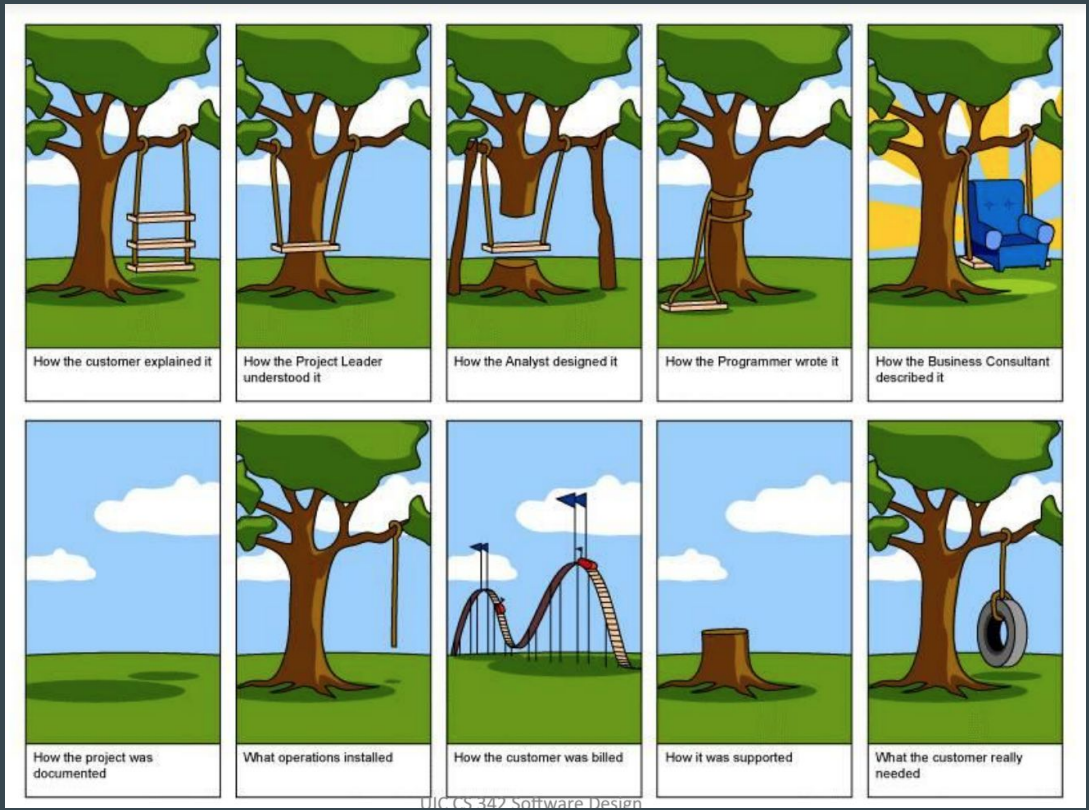
Data Access Layer

Database

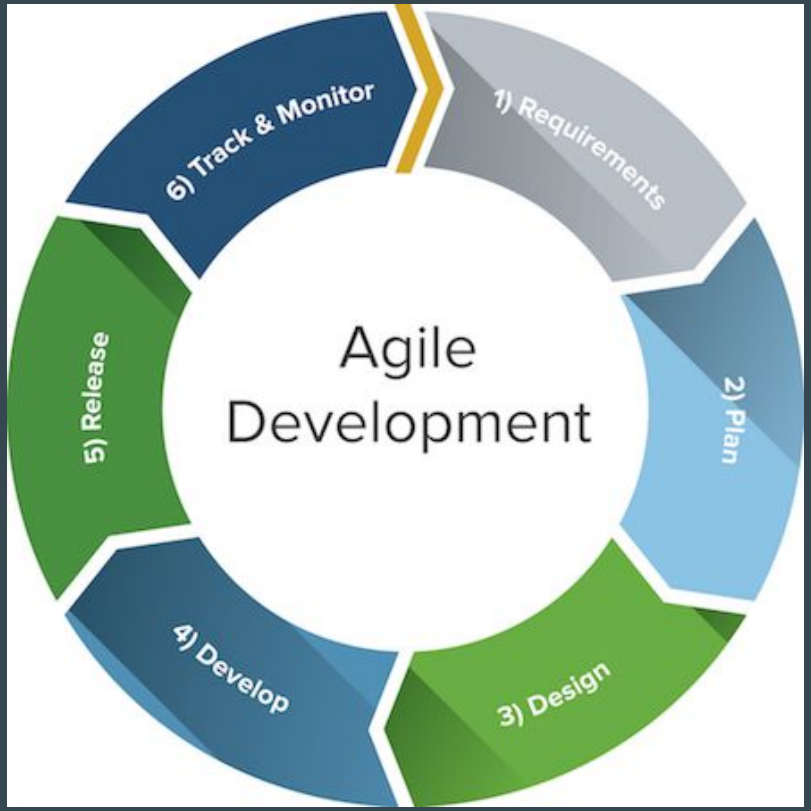
Waterfall model



What's the problem with waterfall?



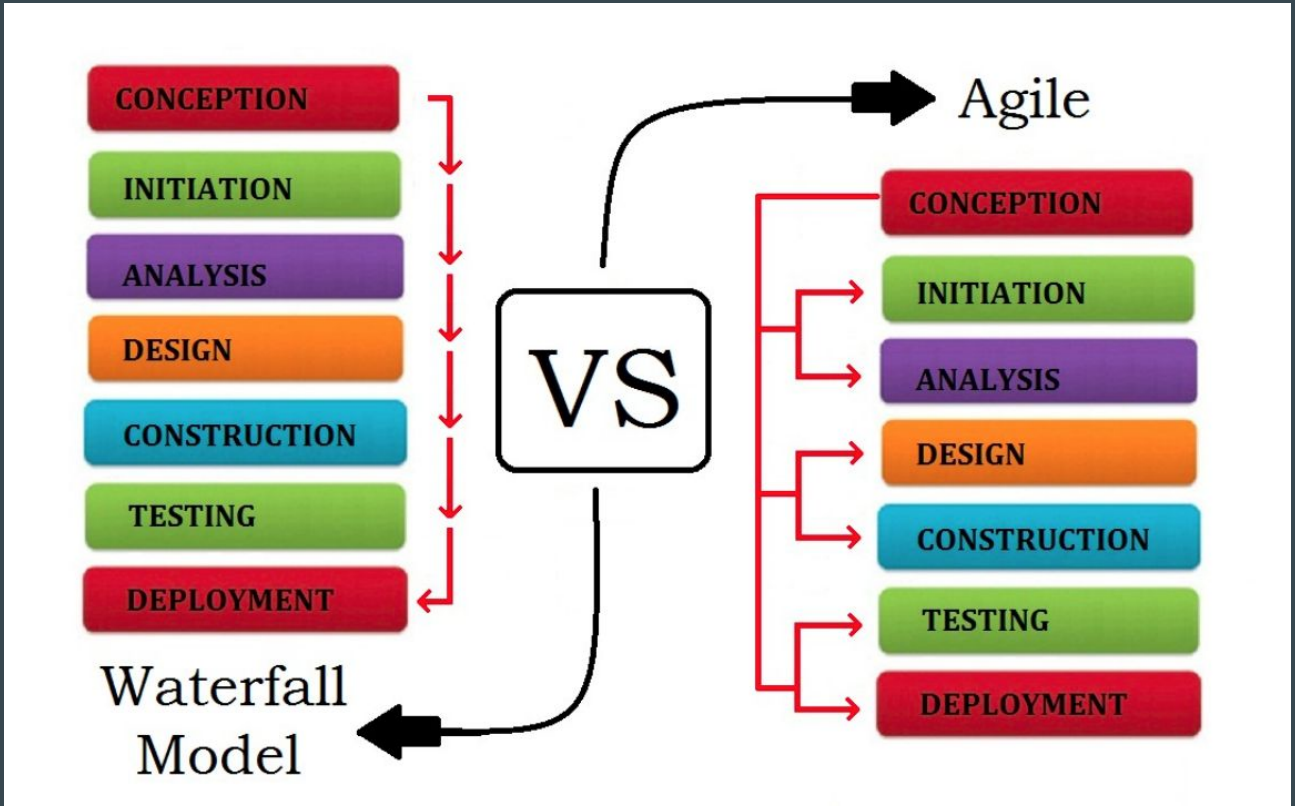
Agile method



Agile method (continued)

- Cross functional team: customer, business analyst, project manager, engineers, QA…
- Present working(ish) software to all stakeholders in iterations and get feedback
- Adaptive
- Input from people with different expertise.
- Advantage and disadvantages

Discussion


CONCEPTION
INITIATION
ANALYSIS
DESIGN
CONSTRUCTION
TESTING
DEPLOYMENT
Waterfall Model
VS
Agile
CONCEPTION
INITIATION
ANALYSIS
DESIGN
CONSTRUCTION
TESTING
DEPLOYMENT

Test driven development (TDD)

Iterative short development cycle

1. Convert spec into an automated test
2. New test will fail at this point…
3. Write minimum code to pass the test
4. Refactor and optimize
5. Repeat

Object oriented programming

- Map SRS into “Object”. Data + operations
- Class vs. object. UIC Students vs. Jacob Huber
- Data Structure vs. Class
- Why OOP?
  - Reuse, flexibility, extensibility, easier to maintain, data protection, intuitive, encapsulation….

Class design

```
1
2  public class Department {
3     private String DepartmentName;
4     private String DepartmentId;
5     //....
6     String [] StudentIds;
7     //...
8     public String GetDepartmentName() {
9        // ...
10    };
11    public int GetTotlal()  {
12       // ...
13    });
14 }
15
16 public class Student {
17    private String NetId;
18    private String FirstName;
19    private String LastName;
20    private StatusEnum Status;//Active, Gradu
21    private Date GraduatetDate;
22    //....
23    public String GetMyDepartmentName() {
24       // ...
25    };
26 }
27
28
29
30
31 public class Sudoku {
```

```
1
2  public class Department {
3     private String DepartmentName;
4     private String DepartmentId;
5     //....
6     public String GetDepartmentName() {
7        // ...
8     };
9     public int GetTotlal()  {
10       // ...
11    });
12 }
13
14 public class Student {
15    private String NetId;
16    private String FirstName;
17    private String LastName;
18    private StatusEnum Status;//Active, Gradudated, DroppedOut....
19    private Date GraduatetDate;
20    private String DepartmentId;
21    //....
22    public String GetMyDepartmentName() {
23       // ...
24    };
25 }
```

Sudoku Solver

- 9x9 grid of 81 squares
- 9 rows, 9 columns or 9 boxes of 3x3 squares
- Fill in the numbers from 1 to 9 so that each row, column and box contain each number from 1 to 9 only once

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Code example

Sudoku.java

```java
public class Sudoku {
    public static void main(String[] args) {
        int[][] matrix = parseProblem(args);
        writeMatrix(matrix);
        if (solve(0,0,matrix))    // solves in place
            writeMatrix(matrix);
        else
            System.out.println("NONE");
    }

    static boolean solve(int i, int j, int[][] cells) {
        if (i == 9) {
            i = 0;
            if (++j == 9)
                return true;
        }
        if (cells[i][j] != 0)  // skip filled cells
            return solve(i+1,j,cells);

        for (int val = 1; val <= 9; ++val) {
            if (legal(i,j,val,cells)) {
                cells[i][j] = val;
                if (solve(i+1,j,cells))
                    return true;
            }
        }
        cells[i][j] = 0; // reset on backtrack
        return false;
    }

    static boolean legal(int i, int j, int val, int[][] cells) {
        for (int k = 0; k < 9; ++k)  // row
            if (val == cells[k][j])
                return false;

        for (int k = 0; k < 9; ++k) // col
            if (val == cells[i][k])
                return false;

        int boxRowOffset = (i / 3)*3;
        int boxColOffset = (j / 3)*3;
        for (int k = 0; k < 3; ++k) // box
            for (int m = 0; m < 3; ++m)
                if (val == cells[boxRowOffset+k][boxColOffset+m])
                    return false;

        return true; // no violations, so it's legal
    }

    static int[][] parseProblem(String[] args) {
        int[][] problem = new int[9][9]; // default 0 vals
        for (int n = 0; n < args.length; ++n) {
            int i = Integer.parseInt(args[n].substring(0,1));
            int j = Integer.parseInt(args[n].substring(1,2));
            int val = Integer.parseInt(args[n].substring(2,3));
            problem[i][j] = val;
        }
        return problem;
    }
```

Reduce searching space by analyzing and reasoning

Each square has 9 possibilities, and there are 81 squares…

- Candidate list reduction
- Single (5, 5)
- Hidden single (3, 7)
- Locked Candidates
- Naked pairs

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Locked Candidate - intersections

2’s in right box can be removed from yellow squares

Naked pairs

6 and 8 should be removed from yellow, so only 1 left

Procedural solution

1. Data structure: 2D array of items
    a. Each element represents a “Square”, a data structure, with candidate values
    b. 9 * 9 array
2. Initialize array from input
3. Reduce candidates by known values
4. Apply “Single”, “hidden single”...
5. Apply “Locked”, “naked pair”
6. Repeat 3...

OOP solution

What are the objects in Soduku problem?

What classes will you create?

What are their methods?