# Welcome to SENG 371
# Software Evolution
# Spring 2013

## A Core Course of the BSEng Program

Hausi A. Müller, PhD PEng
Professor, Department of Computer Science
Associate Dean Research, Faculty of Engineering
University of Victoria

## Announcements

- Final exam
  - Sat, April 13 — 7:00 -10:00 pm
- Last lecture
- No lecture on Monday due to Easter break
  - Thu, April 4
  - Review and wrap-up
- Today
  - AntiPattern plays
  - Turn in eval forms at the end of lecture
- Marking
  - A2 marks are posted
- Assignment 3
  - Due Thu, April 4
  - Part I — Define software evolution terms
  - Part II — Investigate two AntiPatterns — Vendor-Lock-In — Analysis Paralysis
  - Part III — Refactoring in IBM Eclipse and MS Visual Studio and Blob AntiPattern
  - Cite your sources
  - Submit by e-mail to seng371@uvic.ca



## Setting the stage

- Suppose we could turn back time to 1968. It so happens that the first software engineering conference was held in 1968 in Garmisch Partenkirchen in Germany. The term software engineering was coined at that conference.
- Given what you know now about the software industry and everything you learned in this course —and all other university courses — what advice would you give to these pioneers?
- Chances are that the advice you would give to these pioneers then would still be valid today. The premise is that if we follow your advice today, chances are we will be better off in the future.



## Group Assignment
## What advice would you give?

- Address the following topics
  - Software evolution
  - Software engineering education
  - Software architecture
  - Program understanding
- Topic selection
  - Select 2 of these topics
  - Select one topic of your own
- Group assignment
  - Break up in groups of 3-4
  - Select reporter and facilitator
  - Pick three topics (see left)
  - Arrange to meet outside class
  - Facilitator directs discussion and keeps time
  - Reporter records findings and presents the findings to the class (3 mins only!)
  - Turn in three slides with findings in point form for posting
  - Write all students names on each slide
  - Presentation (3 mins only) next Thu

**Please give advice !!**
**You might change history!!**



## Reading Assignment

- Murphy, Notkin, Lan: An empirical study of static call graph extractors, *ACM Transactions on Software Engineering and Methodology (TOSEM)* 7(2):158-191 (1998)
  - http://dl.acm.org/citation.cfm?id=279314
- Müller, Jahnke, Smith, Storey, Tilley, Wong: Reverse Engineering: A Roadmap, in *The Future of Software Engineering*, pp. 47-60 (2000)
  - http://dl.acm.org/citation.cfm?id=336526
- Storey: Theories, tools and research methods in program comprehension: past, present and future, *Software Quality Journal* 14:187-208 (2006)
  - http://webhome.cs.uvic.ca/~chisel/pubs/storey-pc-journal.pdf
- Brown, Malveau, McCormick III, Mowbray: *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley (1998)
- AntiPatterns Tutorial and Website
  - http://www.antipatterns.com/briefing/index.htm
  - http://www.antipatterns.com



## Software AntiPatterns

http://en.wikipedia.org/wiki/The_Comedy_of_Errors

## Group Assignment
## An AntiPattern "Comedy of Errors" (Play)

- Groups of 4 students
- Pick an AntiPattern
- Develop a play to enact the AntiPattern
- Perform the play in class next week
  - Make sure all group members are involved—ideally equally
  - Include props if need be
  - Practice the play (!)
  - 5 mins for play

http://en.wikipedia.org/wiki/The_Comedy_of_Errors

7

## Pick your play to be performed

- Reinvent the Wheel
  - Mon: Morgan, Nic, Vish, Marcelo
- Design By Committee
  - Mon: Michael, Y, Sam, Mackenzie
- Mushroom Management
  - Mon: Daniel, Brad, Dave, George
- Corncob
  - Thu: Geoff, Adam, Scott, Justin
- Golden Hammer
  - Thu: Rob, Ian, Kai, Saleh
- Walking through a Minefield
  - Thu: Jordan, Amanda, Brandon, Romil
- Poltergeists
  - Thu: Curtis, Mikko, Paul, Allan
- Spaghetti Code
  - Thu: Jeremy, Anita, Wes

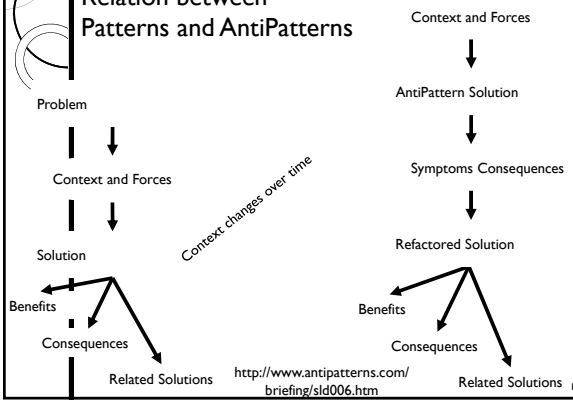
8

SENG 321 — AntiPattern Group Presentations
Evaluation Form

| | | |
|---|---|---|
| Evaluator's name | | |
| | | |
| Group/Project Name: Reinvent the Wheel —Mon: Morgan, Nic, Vish, Marcelo | | |
| | | |
| Quality of presentation | | |
| Well rehearsed | 4 | |
| Props | 2 | |
| Problem clearly described | 4 | |
| Solution clearly described | 4 | |
| Acting performance | 3 | |
| Closing: main points reiterated; strong; positive attitude and outlook | 3 | |
| Subtotal | 20 | |
| Other comments | | |

9

## Relation between Patterns and AntiPatterns

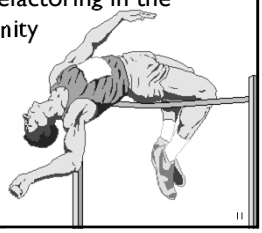



http://www.antipatterns.com/briefing/sld006.htm

10

## Refactoring
## Turn AntiPattern into a Useful Pattern

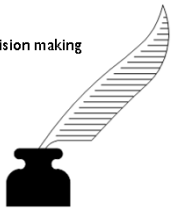
- An approach for evolving the solution into a better one
- This process of change, migration, or evolution is called refactoring in the AntiPattern community



11

## Reference Model

- Root causes
  - provide fundamental context for the AntiPattern
- Primal forces
  - are the key motivators for decision making
- Software design-level model
  - define architectural scales; each pattern has a most applicable scale



12

## Root Causes

- Haste
  - hasty decisions compromise quality
  - code that appears to work is acceptable
  - testing is ignored
- Apathy
  - lack of partitioning
  - ignoring the separation of concerns (e.g., stable vs. replaceable design)

13

## Root Causes …

- Narrow-mindedness
  - refusal of known or accepted solutions
  - reluctance to use metadata
- Sloth
  - poor decision based on an easy answer
  - frequent interface changes
  - lack of configuration control
  - reliance on generating stubs and skeletons

14

## Forces …

- Vertical forces
  - Domain specific
  - Unique to a particular situation
- Horizontal or primal forces
  - Applicable across multiple domains
  - Influence design and reengineering choice across several software modules and components
  - Choices made elsewhere may impact local choices

15

## Primal Forces …

- Management of functionality
  - Meeting the requirements
- Management of performance
  - Meeting required speed and operation
- Management of complexity
  - Defining abstractions
- Management of change
  - Controlling the evolution of the software
- Management of IT resources
  - People and IT artifacts
- Management of technology
  - Controlling technology evolution

16

## AntiPattern ViewPoints

**Gof4 patterns**
Creational
Structural
Behavioral

- Developer
  - Situations encountered by programmers
  - http://www.antipatterns.com/briefing/sld012.htm
- Architect
  - Common problems in system structure
  - http://www.antipatterns.com/briefing/sld014.htm
- Manager
  - Affect people in all software roles
  - http://www.antipatterns.com/briefing/sld016.htm

17

## Software Development AntiPatterns

- The Blob
- Continuous obsolescence
- Lava Flow
- Ambiguous viewpoint
- Functional decomposition
- Poltergeists
- Boat Anchor

18

## Software Development AntiPatterns

- Golden Hammer
- Dead End
- Spaghetti Code
- Input Kludge
- Walking through a Minefield
- Cut-and-Paste Programming
- Mushroom Management

19

## The Blob

- Problem
  - Procedural style design leads to one object with a lion's share of the responsibilities
  - Most other objects only hold data
  - This is the class that is really the heart of our architecture
  - One class monopolizes the processing and the others encapsulate data

20

## The Blob

- Causes
  - Lack of an object-oriented architecture
  - Lack of architecture enforcement
  - Procedural design expert are chief architects
  - Wrapping a legacy system results in a Blob … acceptable

21

## The Blob …

- Solution
  - Distribute responsibilities more uniformly
  - Isolate the effect of changes (encapsulation)
  - Identify or categorize attributes and operations
  - Find "natural homes" for the identified classes
  - Remove outliers

22

## Continuous Obsolescence

- Problem
  - Technology is changing rapidly
  - Developers have difficulty keeping up
  - Product releases don't work together
- Solution
  - Open systems standards
  - Use consortium standards since they represent industry consensus
  - Stable system interfaces to separate concerns
  - Open source

23

## Ambiguous Viewpoint

- Problem
  - Old OOA&D models and methods often do not explain their viewpoint
    - Object oriented analysis and design
  - Often implementation view—least useful
- Solution
  - Provide different viewpoints
  - Separation of concerns
  - Interfaces, db, application code
  - 14 diagram types in UML 2+

24

## Functional Decomposition

- Problem
  - Result of experienced, non-oo developers
  - Procedural design in an oo language
  - Class-based versus object-oriented code
  - Complex and clever code
  - Function-centric design as opposed to balanced approach function/data-centric
- Solution
  - Object-oriented redesign
  - Package data and methods
  - Separation of concerns

25

## Poltergeists

- Problem
  - Classes with limited roles or life cycles
  - Start a process for another object
  - Short-lived objects
- Solution
  - Refactor into longer-lived objects
  - Package data and methods

26

## Boat Anchor

- Problem
  - A piece of software that does not serve a useful purpose on the current project
  - A costly acquisition which management is reluctant to let go
- Solution
  - Ditch the anchor

27

## Golden Hammer

- Problem
  - A familiar and proven technology or concept that is applied obsessively to many software problems (e.g., MVC)
- Solution
  - Expand the knowledge of developers through courses, training, books
  - Expose developers to alternative technologies and approaches

28

## Dead End

- Problem
  - Modifying a reusable component even if it is no longer maintained or supported by the supplier
  - Amount of maintenance increases significantly
  - Dead code
- Solution
  - Outsource maintenance rather than import maintenance

29

## Spaghetti Code

- Problem
  - Most famous AntiPattern
  - Many complexity measure have been invented to assess it
  - Common for programmer who cannot abstract
    - Large interfaces, many parameters,
  - Not very common in industrial projects
    - More of a myth than anything else
- Solution
  - Many automatic tools available

30

## Object-oriented Spaghetti Code

- Many methods with no parameters
- Suspicious class or global variables
- Intertwined and unforeseen relationships among objects
- Process-oriented methods, object with process-oriented names
- Inheritance and polymorphism cannot be used to extend the system

31

## Cute and Paste Programming or Cloning

- Problem
  - Software clones
  - "Hey, I thought you fixed that bug already, so why is it doing this again?"
  - "Wow, you guys work fast. Over 400KLOC in three weeks is amazing!"
  - Degenerate form of reuse
  - Very common in COBOL

32

## Cute and Paste Programming or Cloning

- Solution
  - Clone detection
  - Parameterize types
  - Introduce an additional level of indirection
  - Exploit polymorphism
  - Dynamic schemas

33

## AntiPattern ViewPoints

**Gof4 patterns**
Creational
Structural
Behavioral

- Developer
  - Situations encountered by programmers
  - http://www.antipatterns.com/briefing/sld012.htm
- Architect
  - Common problems in system structure
  - http://www.antipatterns.com/briefing/sld014.htm
- Manager
  - Affect people in all software roles
  - http://www.antipatterns.com/briefing/sld016.htm

34

## Software Architecture AntiPatterns

- Autogenerated Stovepipe
- Stovepipe Enterprise
- Jumble
- Stovepipe System
- Cover Your Assets
- Vendor Lock-in
- Wolf Ticket

35

## Software Architecture AntiPatterns …

- Architecture By Implication
- Warm Bodies
- Design By Committee
- Swiss Army Knife
- Reinvent the Wheel
- The Grand Old Duke of York

36

## Autogenerated Stovepipe

- Problem
  - Migrating an existing system to a distributed, network-centric or Web-services based system
  - Converting existing software interfaces (e.g., functions and classes) to distributed interfaces
  - Existing interfaces use fine-grain data (e.g., parameters)

37

## Autogenerated Stovepipe …

- Solution
  - Reengineer interfaces
  - Make the interfaces larger to fit the "new stove pipe"
  - Define a separate, larger-grain object model
  - The interoperability among subsystems constitutes the core of the new design
  - Aim for stable interfaces; even more important for distributed, network-centric and Web-services based systems than for standalone systems

38

## Stovepipe Enterprise

- Problem
  - Stovepipe Enterprise is characterized by a software structure that inhibits change
  - Must be constantly repaired
  - Changes are done one hole at a time, often patched over
  - Brittle, monolithic system architectures (usually undocumented)
  - Inability of systems to interoperate

39

## Stovepipe Enterprise …

- Solution
  - Force stable interfaces
  - Form product lines
  - Identify requirements for the entire system or enterprise
  - Write specification documents for the entire enterprise
  - Coordination of technologies at several levels
  - Identify common standards and migration direction with a standard reference model
  - Usage conventions across systems
  - Detailed interoperability conventions across systems
  - Data, control and presentation integration standards

40

## Design by Committee

- Problem
  - Gold Plating, Standards Disease, Make Everybody Happy, Political Party
  - Project team is egalitarian; everyone has equal say; decisions are democratic
  - The majority rule leads to diffusion of abstraction and excess complexity
  - "A camel is a horse designed by a committee."

41

## Design by Committee …

- Symptoms
  - Design documentation is voluminous
  - The requirements do not converge and are unstable
  - Design meetings are slow, concentrate on details, and avoid big picture discussions
  - Decisions are only made in meetings
  - No prioritization of design features

42

## Design by Committee …

- Causes
  - No designated project architect
  - Ineffective meeting facilitation
  - The suggestions of all committee members are incorporated to keep everybody happy
  - No separation of concerns

43

## Design by Committee …

- Refactored solution
  - Reform the meeting process
  - Why are we here?
  - What outcomes do we want?
  - Assign explicit roles
    - Owner, facilitator, architect, developer, tester, domain expert
    - "My specialty is being right when other people are being wrong." — George Bernard Shaw

44

## Design by Committee …

- Employ Spitwads meeting process
  - Ask question—How can we improve performance?
  - Write down answer silently
  - Toss spitwads à la Michael Jordan
  - Redistribute, read, and record spitwads
  - Reach common understanding
  - Eliminate duplicates
  - Prioritize by voting
  - Discuss highest priority selections

45

## Design by Committee …

- SQL example
  - SQL89—115 pages
  - SQL92—580 pages
  - SQL3—still not complete; may never be fully implemented; a dumping ground for advanced database features
  - Better solutions
  - Open Database Connectivity (ODBC)
  - Java Database Connectivity (JDBC)

46

## Reinvent the Wheel

- Problem
  - Our problem is unique
  - Developers have minimal knowledge of each other's code
  - Building systems from the ground up even though related legacy systems exist
  - The existence of legacy systems is the norm rather than the exception
  - Lack of program families or product lines

47

## Reinvent the Wheel …

- Symptoms
  - Closed system architectures—no provision of reuse, interoperability, or change management
  - Replication of COTS components
  - Inability to deliver desired features on time and within budget
  - Corporate knowledge is not leveraged

48

## Reinvent the Wheel …

- Causes
  - No communication and technology transfer among software development projects
  - Corporate knowledge is not leveraged
  - No explicit architecture process
  - Lack of enterprise management

49