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
  - Review
  - What I learned in this course
  - What advice would I give considering what I learned
- Assignment 3
  - Due Thu, April 4 — today
  - 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

2

Final—Format

- Closed books, closed notes, no calculators, no gadgets
- 15-20 questions
- Same format as the midterm
- Time should not be a problem
- Attempt all questions !!
- The final is hard ☺

3

Final—Materials

- Everything we discussed in class
- All lecture slides including
  - All slides before midterm
  - All slides after midterm
  - All lab slides:
    - Visualization, CVS, GIT, GITHUB, ANT
    - IEEE Standard
- Midterm
  - Study it!
  - 1-2 final questions are midterm questions
- Three major reading assignments
  - See next slides

4

Reading Assignment I

- IBM Corporation: An Architectural Blueprint for Autonomic Computing, Fourth Edition (2006)
  http://people.cs.kuleuven.be/~danny.weyns/csds/IBM06.pdf
- Truex, Baskerville, Klein: Growing Systems in Emergent Organizations. Communications of the ACM, 42(8):117-123 (1999).
  http://portal.acm.org/citation.cfm?id=310930.310984&coll=GUIDE&dl=GUIDE,ACM&CFID=2240896&CFTOKEN=98671917
- Northrop, et al.: Ultra-Large-Scale Systems. The Software Challenge of the Future. Technical Report, Software Engineering Institute, Carnegie Mellon University, 134 pages ISBN 0-9786956-0-7 (2006)
  http://www.sei.cmu.edu/uls

5

Reading Assignment II

- Chikofsky, Cross: Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software 7(1):13-17 (1990)
  http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=43044
- Kienle, Müller: Rigi—An Environment for Software Reverse Engineering, Exploration, Visualization, and Redocumentation, *Science of Computer Programming* 75(4):247-263, Elsevier, Apr. 2010.
  http://www.sciencedirect.com/science/article/pii/S016764230900149X
- Müller, Jahnke, Smith, Storey, Tilley, Wong, Reverse Engineering: A Roadmap, in *The Future of Software Engineering, ICSE 2000 Millennium Celebration,* 2000.
  http://dl.acm.org/citation.cfm?id=336526

6

Reading Assignment III

- 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

7

Course Review

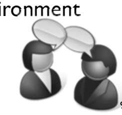
Final exam

Sat, April 13 — 7:00 -10:00 pm

8

Some basic definitions

- *Software* - the programs, documentation, and operating procedures by which computers can be made useful to humans
- *Software evolution* - a process of continuous change from a lower, simpler to a higher, more complex, or better state
- *Software maintenance* - modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment
- *Maintainability* - the ease with which maintenance can be carried out



9

Maintenance versus evolution... (1)

- The term *Software Engineering* was coined in 1968 at a NATO meeting to address the upcoming "software crisis"
- Maintenance was considered to be something that was done after delivery (as in the waterfall model)
- Evolution captures the more realistic evolutionary model of software (it is never "done")
- Manny Lehman in the 70's proposed laws of software evolution, after studying the IBM OS 360 operating system – findings later confirmed in other studies, especially of proprietary systems
- Term gaining more acceptance since the 90's

10

Maintenance versus evolution... (2)

- Software that is used in the real world, will need to adapt as the world continually changes
- Software that doesn't permit change is said to suffer from decay – a poorly degraded system will have to be phased out (sometimes called a legacy system)
- Software evolution is also fundamental in agile development which recognizes the need to continually adapt to changing requirements in a lightweight and agile manner
- Nowadays the terms software evolution and software maintenance are considered synonyms
- Prefer the term evolution, because maintenance may imply that the software has deteriorated in some way

11

Difference between maintenance and "green field development"

- Maintenance is constrained by parameters of existing system
- "Impact analysis" important step in maintenance
- How can the change be accommodated?
- What ripple effects will there be?
- Determine skills and knowledge required to get the job done

12

Analogy from the building industry

"The architect and the builders must take care not to weaken the existing structure when additions are made. Although the costs of the new room usually will be lower than the costs of constructing an entirely new building, the costs per square foot may be much higher because of the need to remove existing walls, reroute plumbing and electrical circuits and take special care to avoid disrupting the current site"

13

Some problems in legacy software maintenance

- Maintenance has a poor image!
- Lack of documentation – especially on "design rationale"
- Architectural decay
- Programmers lacking in domain/ application knowledge
- Unstructured code
- Old code that can't be thrown away (mixed languages, special purpose hardware)

14

Why maintenance is needed?

- To provide continuity of service
- Bug fixing, recovery from failure, change in platform OS, hardware etc.
- To support mandatory upgrades
- Changing laws, regulations, competitive edge
- To support user requests
- To support future maintenance

15

Categorizing software change

- Modification initiated by defects
- Modifications driven by changes to the environment of the system
- Changes undertaken to expand the existing requirements on a system
- Change undertaken to prevent malfunction
- Change undertaken to increase maintainability

16

Types of maintenance

- *Corrective maintenance*: Reactive modification of a software product performed after delivery to correct discovered problems
- *Adaptive maintenance:* Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment
- *Perfective maintenance:* Modification of a software product after delivery to improve performance or maintainability
- *Preventive maintenance:* Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults

17

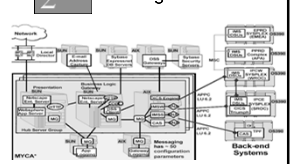
Complexity of configurations

- Application Server
  - ~100 configuration parameters
  - Several applications
  - Hundreds of servlets
  - Tens of EJBs
- Web Server
  - ~20 configuration parameters
  - Serves thousands of web artifacts
- Messaging
  - ~30 configuration parameters
- DBMS, TCP/IP, OS …

Information systems are very complex for humans and costly to install an maintain

x 2-5 parameters

2^{150} settings



18

The Evolution Problem:

Devices, environment, infrastructure, web, services, business goals, user expectations, … all evolve over time

— thus, *software must evolve*

Goal: Trouble Free Systems

Build a system used by millions of people each day administered and managed by a half-time person

— Jim Gray, Microsoft Research

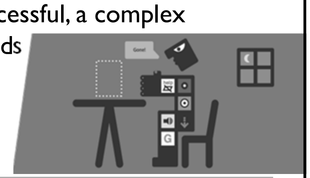
The Complexity Problem

- The increasing complexity of computing systems is overwhelming the capabilities of software developers and system administrators to design, evaluate, integrate, and manage these systems
- Major software and system vendors are concluding that the only viable long-term solution is to create computing systems that manage themselves

… elusive goal?

The Conquest of Complexity

- There has never been anything quite like information technology before, but there have certainly been other complex technologies that needed simplifying
- To be truly successful, a complex technology needs to "disappear"

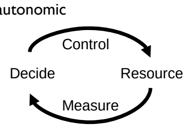


Source: A. Kluth. Information Technology. *The Economist*, Oct 28, 2004

What is Autonomic Computing?

- Webster's definition
  - Acting or occurring involuntarily; automatic: an autonomic reflex
  - Relating to, affecting, or controlled by the autonomic nervous system or its effects or activity
  - Autonomic nervous system: that part of the nervous system that governs involuntary body functions like respiration and heart rate

- IBM's definition
  - An approach to self-managed computing systems with a minimum of human interference
  - The term derives from the body's autonomic nervous system, which controls key functions without conscious awareness or involvement



What autonomic or self-managing systems deliver

Increased Responsiveness

Adapt to dynamically changing environments

Business Resiliency

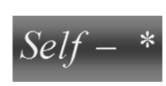
Discover, diagnose, and act to prevent disruptions

Operational Efficiency

Tune resources and balance workloads to maximize use of IT resources

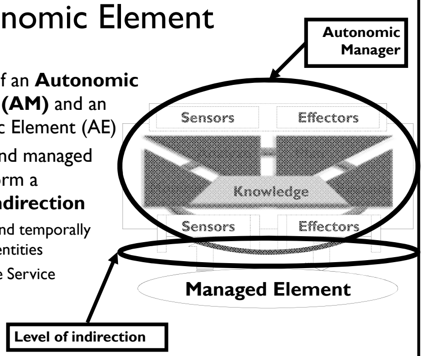
Secure Information and Resources

Anticipate, detect, identify, and protect against attacks



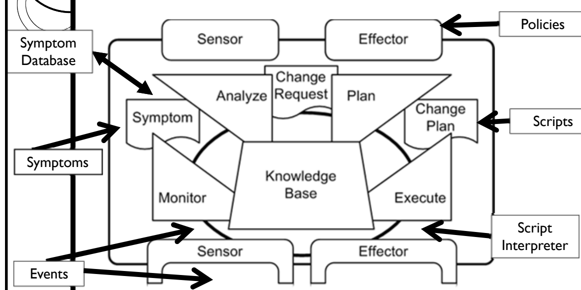
Autonomic Element

- Consists of an **Autonomic Manager (AM)** and an Autonomic Element (AE)
- Manager and managed element form a **level of indirection**
  - Spatially and temporally separate entities
  - Enterprise Service Bus



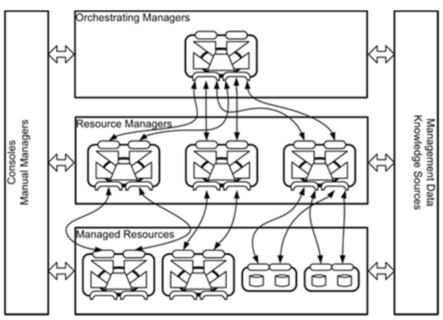
25

MAPE-K Loop Standards & Interfaces



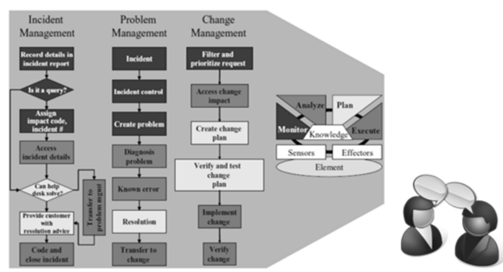
26

ACRA: Autonomic Computing Reference Architecture



27

MAPE-K Loops in IT Processes



IBM: An Architecture Blueprint for Autonomic Computing, 4th Ed. 2006

28

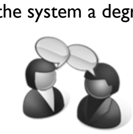
Self-Adaptive Systems My Favourite Definition

- A self-adaptive system continuously adjusts its behaviour at run-time in response to its perception of its environment and its own state in the form of fully or semiautomatic self-adaptation.
- H. Giese, Y. Brun, J. Serugendo, C. Gacek, H. Kienle, H. Müller, M. Pezzè, M. Shaw.: Engineering Self-Adaptive and Self-Managing Systems, LNCS 5527, Springer, 2009.

29

Key Questions

- What aspects of the environment should a self-adaptive system monitor?
  - The system cannot monitor everything in the environment
  - What aspects of the environment are truly relevant?
- How should a self-adaptive system react if it detects changes in the environment?
  - Maintain high-level goals
  - Relax non-critical goals to allow the system a degree of flexibility
  - Goal trade-off analysis



30

Laws of software evolution

1. Law of Continuing Change (1974)
   - "E-type systems must be continually adapted or they become progressively less satisfactory."
   - Software which is used in a real-world environment must change or become less and less useful in that environment.
2. Law of Increasing Complexity (1974)
   - "As an E-type system evolves its complexity increases unless work is done to maintain or reduce it."
   - As an evolving program changes, its structure becomes more complex, unless active efforts are made to avoid this phenomenon.

Laws of software evolution …

3. Law of Self Regulation (1978)
   - "E-type system evolution process is self regulating with distribution of product and process measures close to normal."
   - System attributes such as size, time between releases, and the number of reported errors are approximately invariant for each system release.
4. Law of Conservation of Organisational Stability
   - "The average effective global activity rate in an evolving E-type system is invariant over product lifetime."
   - Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.

Laws of software evolution …

5. Law of Conservation of Familiarity (1978)
   - "As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery."
   - Over the lifetime of a system, the incremental system change in each release is approximately constant.
   - The average incremental growth of systems tends to remain constant or decline over time.
6. Law of Continuing Growth (1991)
   - "The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime."
   - Functional capability must increase over the lifetime of a system to maintain user satisfaction.

Laws of software evolution …

7. Law Declining Quality (1996)
   - "The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes."
   - Unless rigorously adapted, quality will appear to decline over time.
8. Law of Feedback System (1996)
   - "E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base"
   - Evolution systems are multi-level, multi-agent, multi-loop feedback systems.

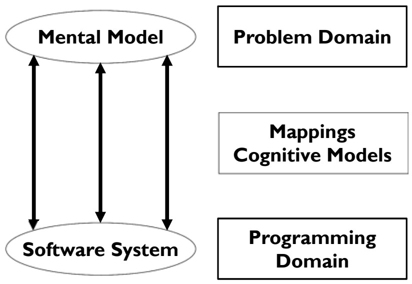
Program understanding

- What strategies do you follow when trying to understand a program written by someone else?
- Describe the kinds of information you use to arrive at an understanding of how it works.

Program comprehension theories and models

- Program comprehension models
  - Bottom up
  - Top down
  - Integrated meta-model
  - Opportunistic, Systematic etc.
- Theories about tool support
  - Cognitive support
  - Improving flow

Developing mental models using cognitive models



Software AntiPatterns—Overview

- Motivation
- Reference model
- Software Development AntiPatterns
- Software Architecture AntiPatterns
- Software Management AntiPatterns
- Summary

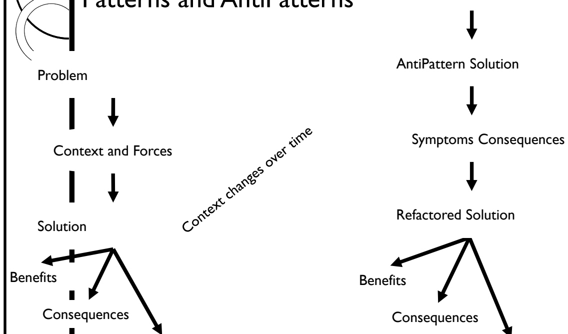
Essence of an AntiPattern

- Two solutions instead of a problem and a solution
  - Problematic solution which generates negative consequences
  - Refactored solution, a method to resolve and reengineer the AntiPattern
- A pattern in an inappropriate context

Relation between Patterns and AntiPatterns

- Design patterns often evolve into an AntiPattern
- Procedural programming was a great design pattern in the 60's and 70's
- Today it is an AntiPattern
- Object-oriented programming is today a practiced pattern ...

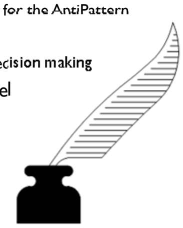
Relation between Patterns and AntiPatterns



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

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



Root causes

- Haste
- Apathy
- Narrow-mindedness
- Sloth
- Avarice
- Ignorance
- Pride

43

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

44

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

45

Summary

- AntiPatterns are normal
- Some AntiPatterns must be tolerated
  - Accept those things you cannot change
  - Have the courage to change those things you can and the wisdom to know the difference. —Serenity Prayer
- Avoid the use of the Golden Hammer
  - Excessive use of one pattern
  - More than 200 well-documented software patterns
    - 23 GoF
    - 17 Buschmann
    - 72 analysis
    - 38 CORBA
    - 42 AntiPatterns
- Consider a range of solutions

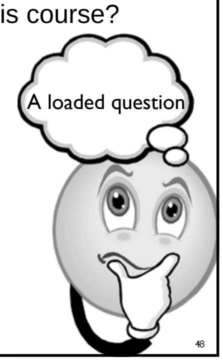
46

Summary …

- During maintenance and evolution one should be particularly aware of the potential presence of AntiPatterns
- Awareness of AntiPatterns is critical for reengineering projects and makes you a better software engineer
- Consider AntiPatterns next time you sign on to a new project
- Invest in reading the AntiPatterns book and web sites

47

What advice would I give to software pioneers considering what I know now and what I learned in this course?



48

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.

49

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!!

50

Congratulations You Arrived!

Have a wonderful summer!

51