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

- Marking
  - Midterm will be returned on Thu in class
  - A1 graded
  - Mon office hours reserved for marking questions —1:30-2:30 ECS 660
- Course website
  - http://www.engr.uvic.ca/~seng371
  - Lecture notes posted
  - Lab slides and activities are posted
- Assignment 2
  - Due March 11 — revised
  - Reverse engineering and program understanding
    - Part I—Summarize three papers
    - Part II—Define terms
    - Part III—Reverse engineer a C program (gawk)
    - Rigi demo on Monday
  - Cite your sources
  - Submit by e-mail to seng371@uvic.ca

Video of the Week

https://www.youtube.com/watch?feature=player_embedded&v=nKIu9yen5nc

Every student in every school should have the opportunity to learn to code...

At the 2:21 mark on the video, you will see an iPad interface for a sound mixer written by UVic alumni at LOUD technologies(Acuma Labs) in Victoria. (http://www.mackie.com/products/dlseries/)

Midterm Question 1 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

Midterm Question 2 Scale Changes Everything

- Characteristics of ULS systems arise because of their scale
  - Decentralization
  - Inherently conflicting, unknowable, and diverse requirements
  - Continuous evolution and deployment
  - Heterogeneous, inconsistent, and changing elements
  - Erosion of the people/system boundary
  - Normal failures
  - New paradigms for acquisition and policy

These characteristics may appear in today's systems, but in ULS systems they dominate.
These characteristics undermine the assumptions that underlie today's software engineering approaches.

Midterm Question 2

ULS Systems Operate More Like Cities

- Built or conceived by many individuals over long periods of time (Rome)
- The form of the city is not specified by requirements, but loosely coordinated and regulated—zoning laws, building codes, economic incentives (change over time)
- Every day in every city construction is going on, repairs are taking place, modifications are being made—yet, the cities continue to function
- ULS systems will not simply be bigger systems: they will be interdependent webs of software-intensive systems, people, policies, cultures, and economics

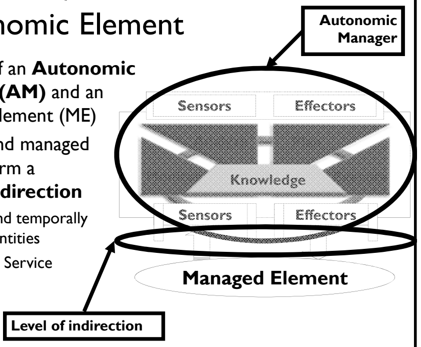
Midterm Question 2 Decentralized Ecosystems

- For 40 years we have embraced the traditional centralized engineering perspective for building software
  - Central control, top-down, tradeoff analysis
- Beyond a certain complexity threshold, traditional centralized engineering perspective is no longer sufficient and cannot be the primary means by which ultra-complex systems are made real
  - **Firms** are engineered—but the structure of the **economy** is not
  - The protocols of the **Internet** were engineered—but not the **Web** as a whole
- **Ecosystems** exhibit high degrees of complexity and organization—but not necessarily through engineering

7

Midterm Question 3 Autonomic Element

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



8

Monitor	**Analyzer**
• Senses the managed process and its context • Collects data from the managed resource • Provides mechanisms to aggregate and filter incoming data stream • Stores relevant and critical data in the knowledge base or repository for future reference.	• Compares event data against patterns in the knowledge base to diagnose symptoms and stores the symptoms • Correlates incoming data with historical data and policies stored in repository • Analyzes symptoms • Predicts problems

MAPE-K Loop
Midterm Question 3

9

Planner	**Execute Engine**
• Interprets the symptoms and devises a plan • Decides on a plan of action • Constructs actions ◦ building scripts • Implements policies • Often performed manually	• Executes the change in the managed process through the effectors • Perform the execution plan • Often performed manually

MAPE-K Loop
Midterm Question 3

10

Midterm question 4 Definitions

- ***Ecosystem***
  - In biology, an ecosystem is a community of plants, animals, and microorganisms that are linked by energy and nutrient flows interacting with each other and with the physical environment.
  - Rain forests, deserts, coral reefs, grasslands, and a rotting log are all examples of ecosystems
- ***Socio-technical ecosystem***
  - An ecosystem whose elements are groups of people together with their computational and physical environments
  - ULS systems can be characterized as *socio-technical ecosystems*
- ***ULS system***
  - A system whose dimensions are of such a scale that constructing the system using development processes and techniques prevailing at the start of the 21st century is problematic.
  - ULS system characteristics
    - Decentralization
    - Conflicting, unknowable, and diverse requirements
    - Continuous evolution and deployment
    - Heterogeneous and changing element
    - Erosion of the people/system boundary
    - Normal failures of parts of the system

cf. Glossary in ULS Book

11

Midterm Question 4 Evolution of Software Systen

- Legacy systems
- Systems of Systems

Ultra-Large-Scale (ULS) Systems
Socio-Technical Ecosystems

12

Midterm Question 4
Change of Perspective

- From satisfaction of requirements through traditional, top-down engineering

The system shall do this
… but it may do this …
as long as it does this …

- To satisfaction of requirements by regulation of complex, decentralized systems

How? With adaptive systems and feedback loops ☺

13

Midterm Question 4
Socio-Technical Ecosystems

- **Socio-technical ecosystems include people, organizations, and technologies at all levels with significant and often competing interdependencies.**
- In such systems there is
  - Competition for resources
  - Organizations and participants responsible for setting policies
  - Organizations and participants responsible for producing ULS systems
  - Need for local and global indicators of health that will trigger necessary changes in policies and in element and system behavior

14

Midterm Question 5
Self-Adaptive Systems

- 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.

15

Midterm Question 5
Key Questions

- How often should adaptation be considered?
  - Policies range from continuous (proactive) adaptation to as-and-when necessary (reactive)
  - Adaptation can also be opportunistic—exploiting resources such as CPU time when it is not being used for other tasks
  - "Go green" adaptation
- What kind of information must be collected to make adaptation decisions
  - Data can be gathered continuously
    - This provides precise and up-to-date observations, but incurs relatively high cost
  - Data can be gathered less often with the resulting samples being approximations of environment activity; this approach imposes less overhead
  - Trust issues

16

Midterm Question 5
Key Questions

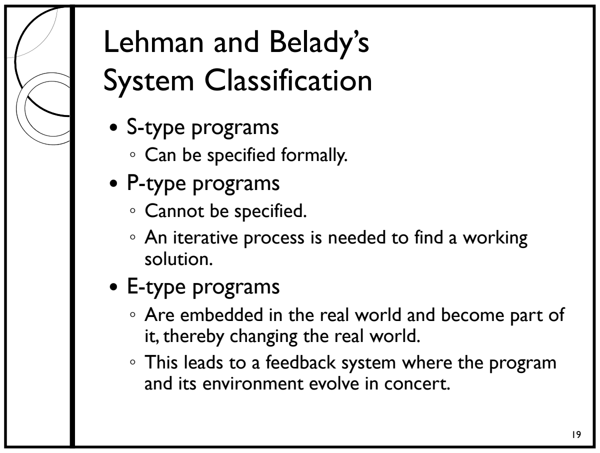
- Under what circumstances is adaptation cost-effective?
- The benefits gained from making a change must outweigh the costs associated with making the change
- Costs include:
  - Performance and memory overhead of monitoring system behaviour
    - Monitoring is necessary to make adaptation decisions
    - Memory may be limited on, particularly if adaptive software runs on embedded devices
  - Decision making—interpreting data gathered from monitoring may be computationally expensive
  - Executing the actions to actually change a system configuration
    - Changes involving physically distributed systems must be coordinated which itself incurs additional overhead

17

Reading assignments

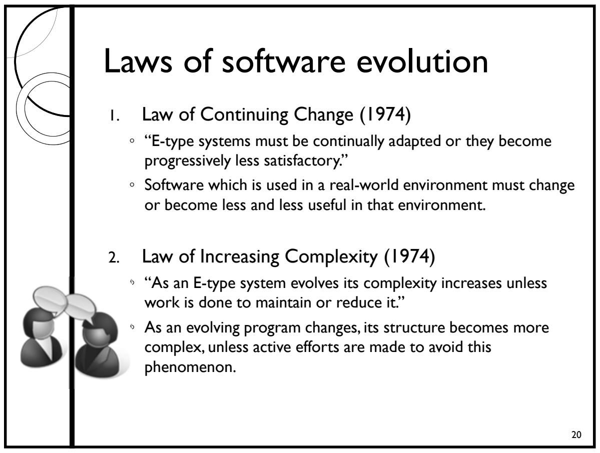
- Chikofsky, Cross: Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software 7(1):13-17 (1990)
  http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=43044
- Kienle, Müller: Rigi—An Environment for Software Reverse Engineering, Exploration, Visualization, and Redocumentation, *Science of Computer 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

18



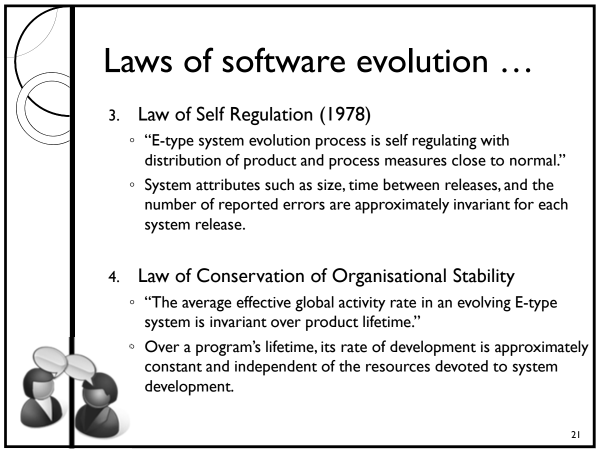
Lehman and Belady's System Classification

- S-type programs
  - Can be specified formally.
- P-type programs
  - Cannot be specified.
  - An iterative process is needed to find a working solution.
- E-type programs
  - Are embedded in the real world and become part of it, thereby changing the real world.
  - This leads to a feedback system where the program and its environment evolve in concert.



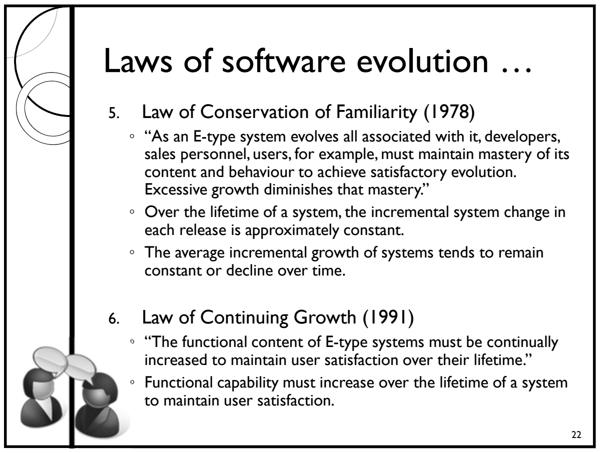
Laws of software evolution

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.



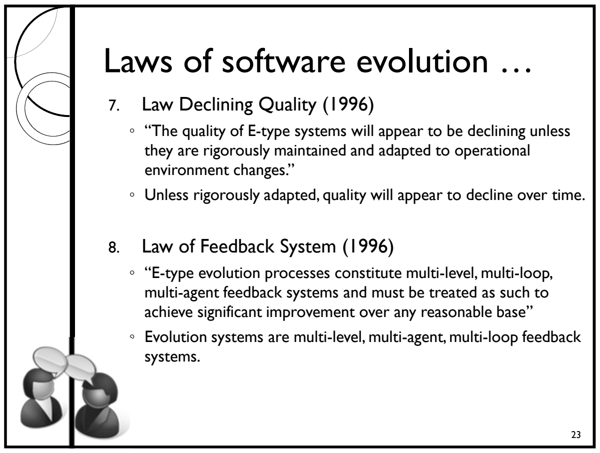
Laws of software evolution …

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.



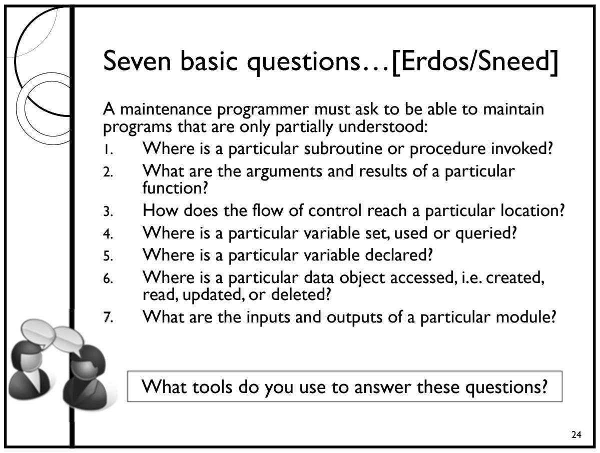
Laws of software evolution …

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.



Seven basic questions…[Erdos/Sneed]

A maintenance programmer must ask to be able to maintain programs that are only partially understood:

1. Where is a particular subroutine or procedure invoked?
2. What are the arguments and results of a particular function?
3. How does the flow of control reach a particular location?
4. Where is a particular variable set, used or queried?
5. Where is a particular variable declared?
6. Where is a particular data object accessed, i.e. created, read, updated, or deleted?
7. What are the inputs and outputs of a particular module?

What tools do you use to answer these questions?

Learning objectives

- Understand differences between reverse engineering, forward engineering and reengineering
- Learn the concepts of design discovery/recovery and re-documentation
- Discuss the application of reverse engineering techniques to software maintenance problems
- Understand the weaknesses in reverse engineering techniques
- Learn about different tools to support reverse engineering

25

Software reverse engineering

- **Def**. A two-step process
  - Information extraction
  - Information abstraction
- **Def**. A three-step process [Tilley95]
  - Information gathering
  - Knowledge organization
  - Information navigation, analysis, and presentation
- **Def**. Analyzing subject system [CC90]
  - to identify its current components and their dependencies
  - to extract and create system abstractions and design information
- The subject system is not altered; however, additional knowledge about the system is produced

26

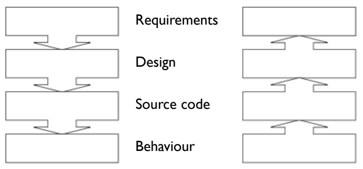
Software reverse engineering …

- Feedback loops in life cycle models (e.g., waterfall or spiral model) are opportunities for reverse engineering
- Related terms
  - Abstraction and composition
  - Design recovery [Big89] and concept assignment [BMW94]
  - Redocumentation [WTMS95]
  - Inverse engineering [RBCM91]
  - Static and dynamic analysis
  - Summarizing resource flows and software structures
  - Change and impact analysis
  - Maintainability analysis
  - Migration analysis
  - Portfolio analysis
  - Economic analysis

27

Forward engineering

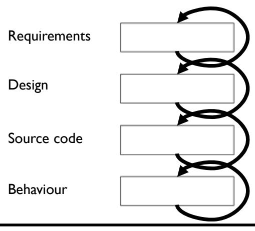
- Traditional software process of moving from high-level abstractions and logical implementation-independent designs to the physical implementation of a system



28

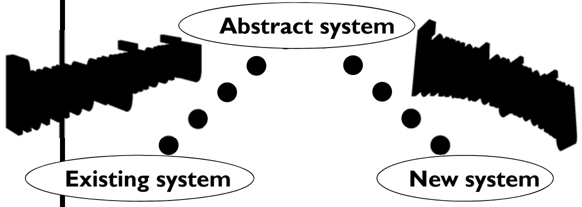
Restructuring

- Transformation from one representation to another at the same relative abstraction level, while preserving the subject's system external behavior



29

The Horseshoe Model of Software Migration



30