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
  - Thu 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—Cite your sources
    - Submit by e-mail to seng371@uvic.ca

Reading assignments
- Chikofsky, Cross: Reverse Engineering and Design
  http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=41044
- Kienle, Müller: Rigi—An Environment for Software
  Reverse Engineering, Exploration, Visualization, and
  Redocumentation, Science of Computer Programming
- Müller, Jahnke, Smith, Storey, Tilley, Wong, Reverse
  Engineering: A Roadmap, in The Future of Software
  http://dl.acm.org/citation.cfm?id=336526
- 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.

IBM OS360/370 Case Studies
- The laws of software evolution were originally based on
  observations regarding the evolution of IBM’s OS/360
  and OS/370.
- The laws were not presented as laws of nature, but
  rather as general observations that are expected to
  hold for all E-type systems, regardless of specific
  programming or management practices.
  - http://www.doc.ic.ac.uk/~mml/

Laws of software evolution
   - “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.
   - “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 …

   − “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.

   − “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.

   − “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.

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.

   − “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
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