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

- Mon, Feb 11
  - Family Day — no class
- Final Exam Date (preliminary)
  - Sat, April 13 — 7:00-10:00 pm
- Course website
  - http://www.engr.uvic.ca/~seng371
  - Lecture notes posted
  - Lab slides and activities are posted
- Assignment 2
  - Due Feb 28
  - Reverse engineering and program understanding
    - Part I—Summarize three papers
    - Part II—Define terms
    - Part III—Reverse engineer a C program (Unix gawk)
  - Cite your sources
  - Submit by e-mail to seng371@uvic.ca

Midterm

- Thu, Feb 14
  - In class, closed books, closed notes
  - All lecture and lab materials covered so far including today
- Topics
  - Definitions: Software evolution, software maintenance, …
  - Software complexity
  - Autonomic systems: autonomic element, autonomic manager, MAPE-K loop, autonomic reference architecture, control loop
  - ULS systems: characteristics, ULS book, web as an example, city as an example, …
  - Self-adaptive and self-managing systems

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

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.

Change of Perspective

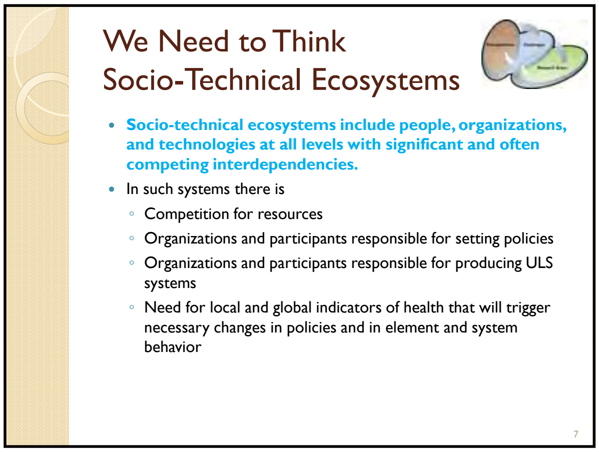
- From satisfaction of requirements through traditional, top-down engineering

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

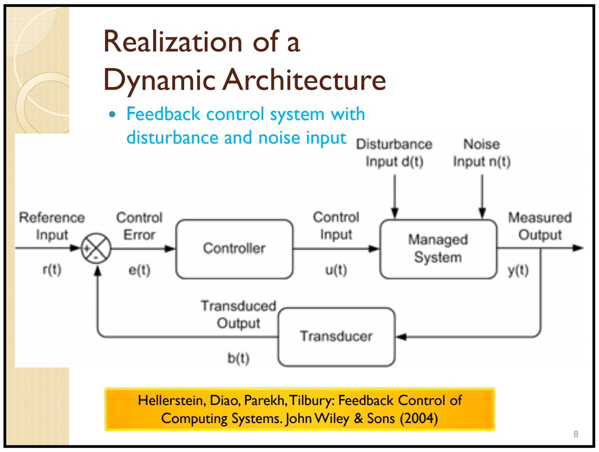
How?

With adaptive systems and feedback loops ☺



We Need to Think 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



Realization of a Dynamic Architecture

- Feedback control system with disturbance and noise input

Disturbance Input d(t)

Noise Input n(t)

Reference Input

r(t)

Control Error

e(t)

Controller

Control Input

u(t)

Managed System

Measured Output

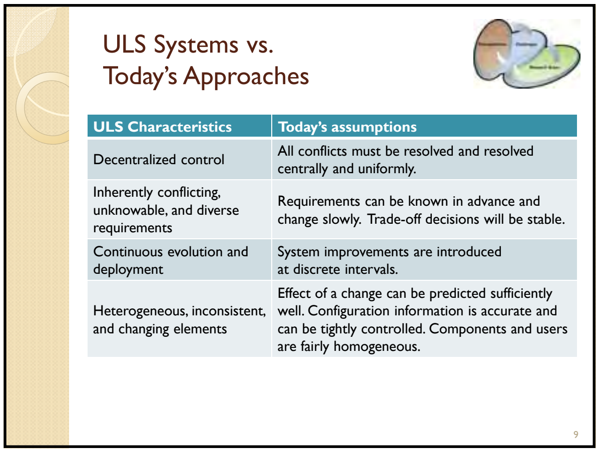
y(t)

Transduced Output

b(t)

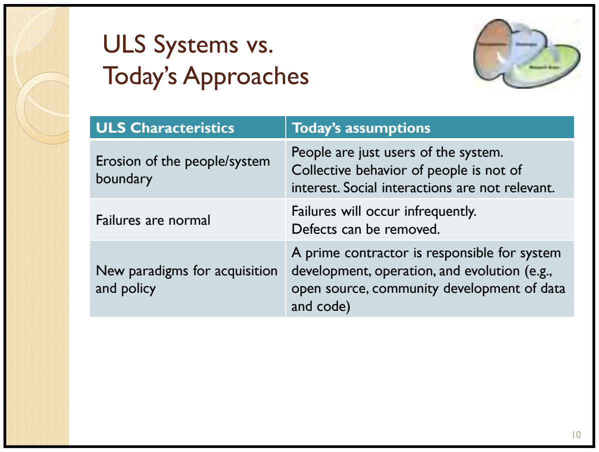
Transducer

Hellerstein, Diao, Parekh, Tilbury: Feedback Control of Computing Systems. John Wiley & Sons (2004)



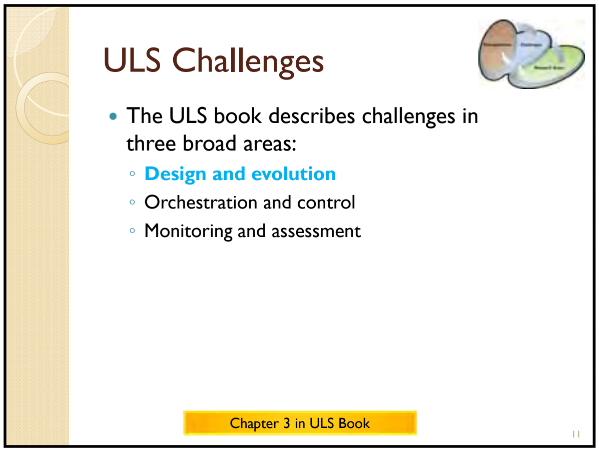
ULS Systems vs. Today's Approaches

ULS Characteristics	Today's assumptions
Decentralized control	All conflicts must be resolved and resolved centrally and uniformly.
Inherently conflicting, unknowable, and diverse requirements	Requirements can be known in advance and change slowly. Trade-off decisions will be stable.
Continuous evolution and deployment	System improvements are introduced at discrete intervals.
Heterogeneous, inconsistent, and changing elements	Effect of a change can be predicted sufficiently well. Configuration information is accurate and can be tightly controlled. Components and users are fairly homogeneous.



ULS Systems vs. Today's Approaches

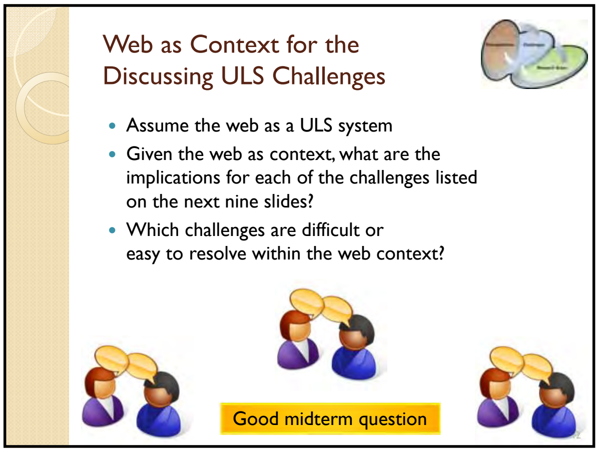
ULS Characteristics	Today's assumptions
Erosion of the people/system boundary	People are just users of the system. Collective behavior of people is not of interest. Social interactions are not relevant.
Failures are normal	Failures will occur infrequently. Defects can be removed.
New paradigms for acquisition and policy	A prime contractor is responsible for system development, operation, and evolution (e.g., open source, community development of data and code)



ULS Challenges

- The ULS book describes challenges in three broad areas:
  - **Design and evolution**
  - Orchestration and control
  - Monitoring and assessment

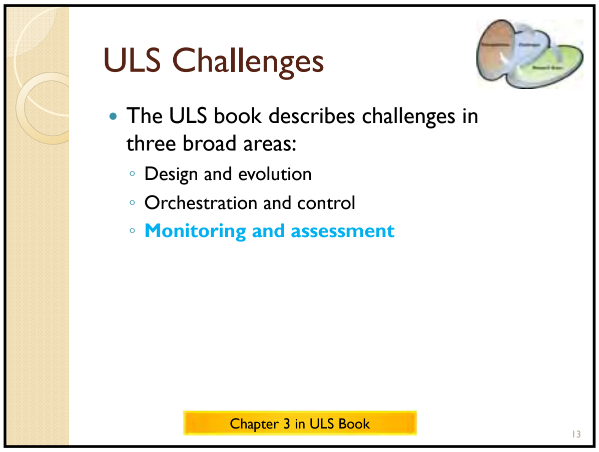
Chapter 3 in ULS Book



Web as Context for the Discussing ULS Challenges

- Assume the web as a ULS system
- Given the web as context, what are the implications for each of the challenges listed on the next nine slides?
- Which challenges are difficult or easy to resolve within the web context?

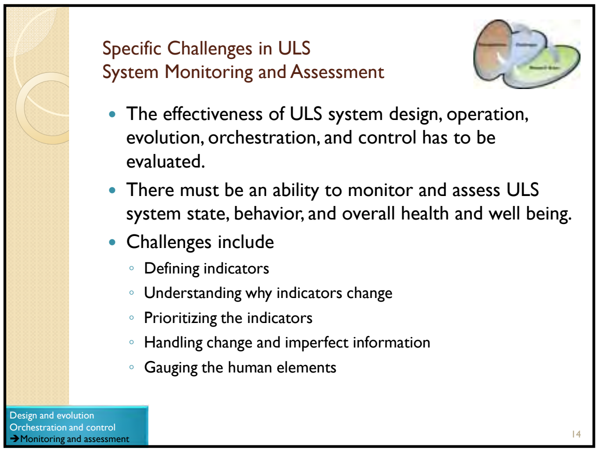
Good midterm question



ULS Challenges

- The ULS book describes challenges in three broad areas:
  - Design and evolution
  - Orchestration and control
  - **Monitoring and assessment**

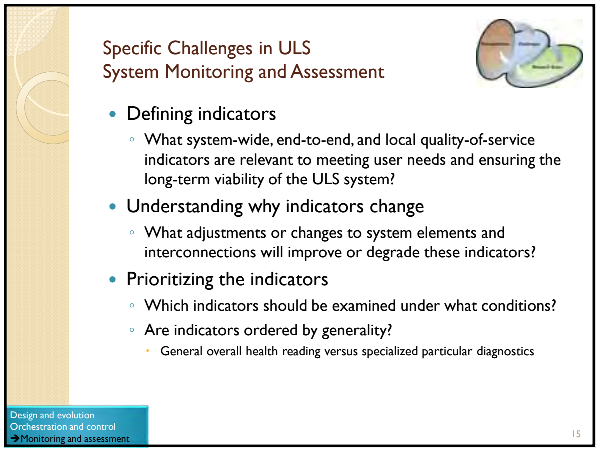
Chapter 3 in ULS Book



Specific Challenges in ULS System Monitoring and Assessment

- The effectiveness of ULS system design, operation, evolution, orchestration, and control has to be evaluated.
- There must be an ability to monitor and assess ULS system state, behavior, and overall health and well being.
- Challenges include
  - Defining indicators
  - Understanding why indicators change
  - Prioritizing the indicators
  - Handling change and imperfect information
  - Gauging the human elements

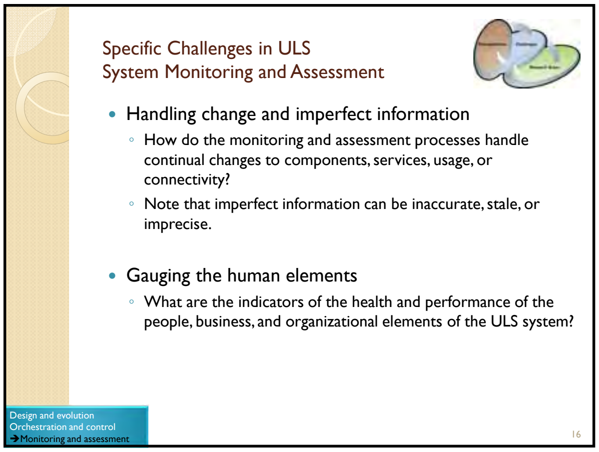
Design and evolution
Orchestration and control
→Monitoring and assessment



Specific Challenges in ULS System Monitoring and Assessment

- Defining indicators
  - What system-wide, end-to-end, and local quality-of-service indicators are relevant to meeting user needs and ensuring the long-term viability of the ULS system?
- Understanding why indicators change
  - What adjustments or changes to system elements and interconnections will improve or degrade these indicators?
- Prioritizing the indicators
  - Which indicators should be examined under what conditions?
  - Are indicators ordered by generality?
    - General overall health reading versus specialized particular diagnostics

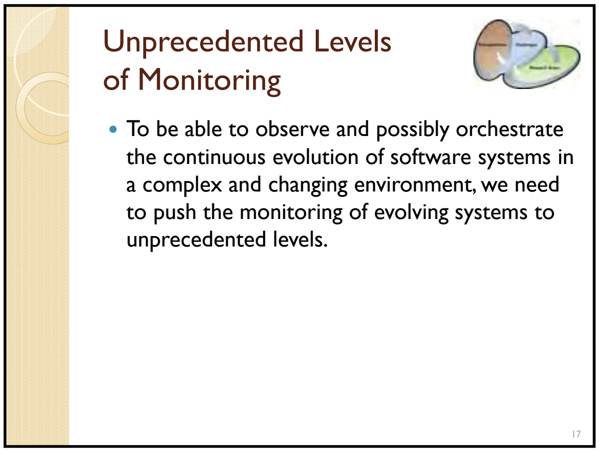
Design and evolution
Orchestration and control
→Monitoring and assessment



Specific Challenges in ULS System Monitoring and Assessment

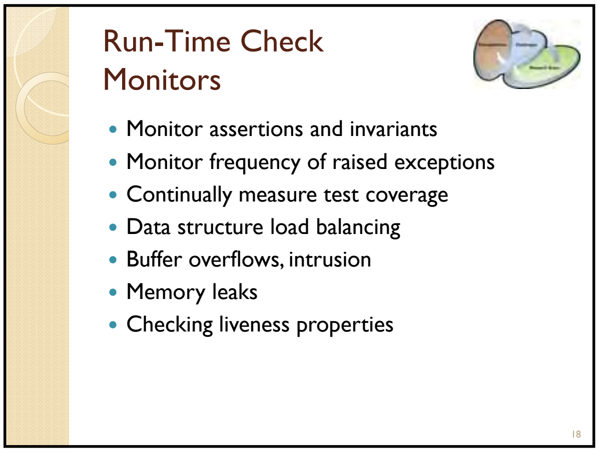
- Handling change and imperfect information
  - How do the monitoring and assessment processes handle continual changes to components, services, usage, or connectivity?
  - Note that imperfect information can be inaccurate, stale, or imprecise.
- Gauging the human elements
  - What are the indicators of the health and performance of the people, business, and organizational elements of the ULS system?

Design and evolution
Orchestration and control
→Monitoring and assessment



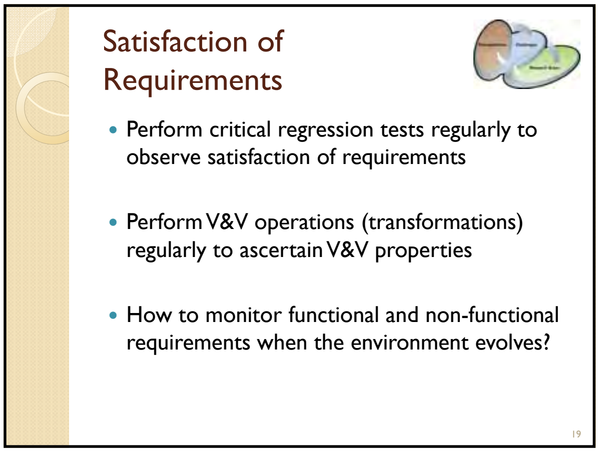
Unprecedented Levels of Monitoring

- To be able to observe and possibly orchestrate the continuous evolution of software systems in a complex and changing environment, we need to push the monitoring of evolving systems to unprecedented levels.



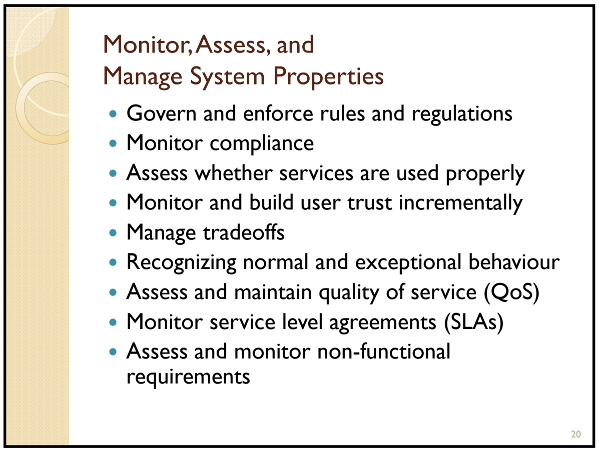
Run-Time Check Monitors

- Monitor assertions and invariants
- Monitor frequency of raised exceptions
- Continually measure test coverage
- Data structure load balancing
- Buffer overflows, intrusion
- Memory leaks
- Checking liveness properties



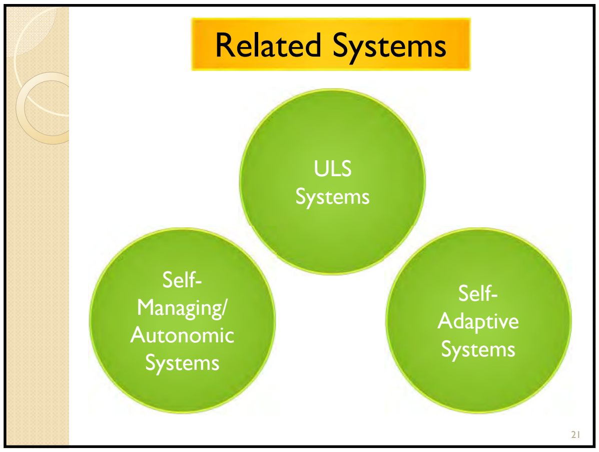
Satisfaction of Requirements

- Perform critical regression tests regularly to observe satisfaction of requirements
- Perform V&V operations (transformations) regularly to ascertain V&V properties
- How to monitor functional and non-functional requirements when the environment evolves?



Monitor, Assess, and Manage System Properties

- Govern and enforce rules and regulations
- Monitor compliance
- Assess whether services are used properly
- Monitor and build user trust incrementally
- Manage tradeoffs
- Recognizing normal and exceptional behaviour
- Assess and maintain quality of service (QoS)
- Monitor service level agreements (SLAs)
- Assess and monitor non-functional requirements

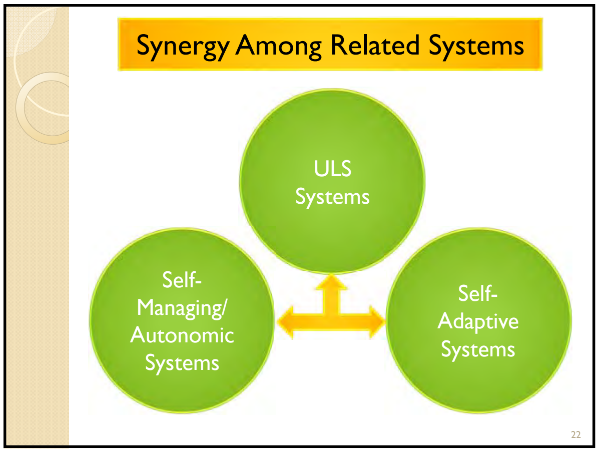


Related Systems

ULS Systems

Self-Managing/ Autonomic Systems

Self-Adaptive Systems

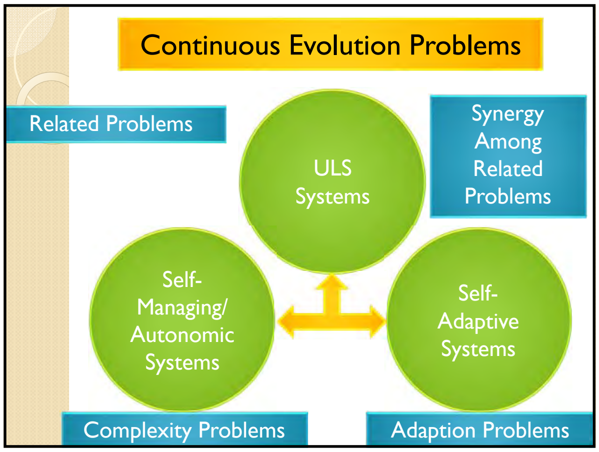


Synergy Among Related Systems

ULS Systems

Self-Managing/ Autonomic Systems

Self-Adaptive Systems



Continuous Evolution Problems

Related Problems

ULS Systems

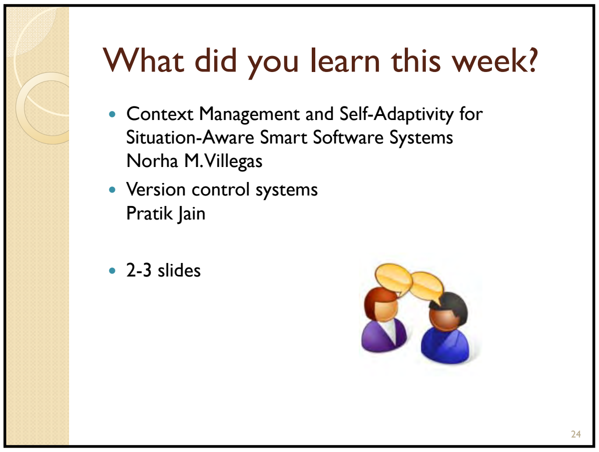
Synergy Among Related Problems

Self-Managing/ Autonomic Systems

Self-Adaptive Systems

Complexity Problems

Adaption Problems



What did you learn this week?

- Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems
  Norha M. Villegas
- Version control systems
  Pratik Jain
- 2-3 slides