Welcome to SENG 371 Software Evolution Spring 2013

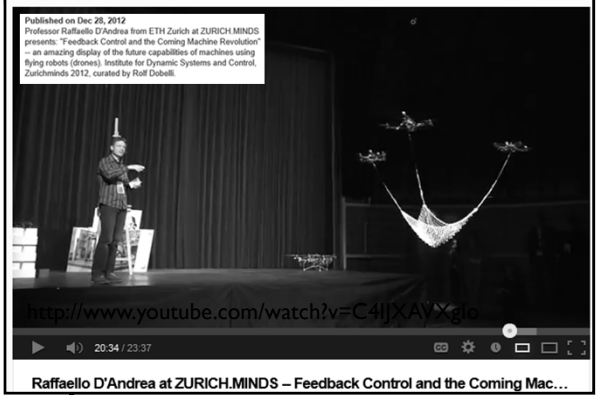
A Core Course of the BSEng Program

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

Announcements

- Lab attendance
  - Has been a problem as of late — needs to change
  - Several questions on labs on final exam
- Final exam
  - Sat, April 13 — 7:00 -10:00 pm
- Marking
  - Midterm and A1 graded
  - Marks posted
- Course website
  - http://www.engr.uvic.ca/~seng371
  - Lecture notes posted
  - Lab slides and activities are posted
- Assignment 2
  - Due March 11 — revised — today midnight
  - Reverse engineering and program understanding
    - Part I—Summarize three papers
    - Part II—Define terms
    - Part III—Reverse engineer a C program (gawk)
  - Cite your sources
  - Submit by e-mail to seng371@uvic.ca
- Assignment 3
  - Will be posted by Thursday

Feedback Control and the Coming Machine Revolution



What were the key points of Raffaello D'Andrea's Keynote?

Views

- 2009—UML 2.2
  - http://www.omg.org/spec/UML/2.2/
  - Seven structural modeling diagrams
  - Seven behavioral modeling diagrams
- 2003—SEI Views
  - Documenting Software Architectures by Clemens, Bachmann, Bass, Garlan, Little, Nord, Stafford
  - http://www.sei.cmu.edu/ata/C4ISR_03/C4ISR_03_1.htm
- 2003—UML 2.0
- 2000—Siemens Views
  - Applied Software Architecture
  - by Hofmeister, Nord and Soni, Siemens
- 1997—UML 1.0
- 1995—Rational Views
  - Also referred to as 4+1 view model of software architecture
  - by Kruchten, Rational
- 1980 Software Cost Reduction (SCR) Method
  - by Parnas et al.
  - Module view, Uses view, Process View
- Programming languages as design notation

An architect must consider a system in at least three ways

- How is the system structured as a set of code units?
- How is the system structured as set of elements that have run-time behaviour and interactions?
- How does the system relate to non-software structures in its environment?

Good final exam question

SEI Views

- Documenting Software Architecture: Views and Beyond
  - Clements, Bachmann, Bass, Garlan, Little, Nord, Stafford, 3rd edition, Sep 2003.
- Module viewtype
  - sSystem structured as a set of code units
  - Document a system's principals units of implementations
- Component-and-connector viewtype
  - System structured as set of elements that have run-time behaviour and interactions
  - Document the system's units of execution
- Allocation viewtype
  - System relate to non-software structures in its environment
  - Document the relationship between a system's software and its development and execution environment

7

Seven rules for sound documentation

- Write documentation from the reader's point of view
- Avoid unnecessary repetition
- Avoid ambiguity
- Use a standard of organisation
- Record rationale
- Keep documentation current but not too current
- Review documentation for fitness of purpose

8

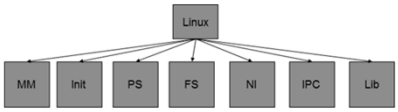
Module viewtype

- A code unit that implements a set of responsibilities
  - Class, collection of classes, layer, or any decomposition of the above (e.g., function decomposition)
  - Properties
    - Responsibilities, visibility information, authors
  - Relations
    - Part of, inherits from (i.e., generalization/spcialization)
  - Styles
    - Decomposition (i.e., subsystem decomposition, Rigi overview window)

9

Linux module view

- Linux subsystems
  - Process Scheduler (PS) – responsible for supporting multitasking by deciding which user process executes.
  - Memory Manager (MM) – provides a separate memory space for each user process.
  - File System (FS)– provides access to hardware devices
  - Network Interface (NI)– encapsulates access to network devices



10

Styles of module viewtype

- Decomposition style
  - Part of hierarchy
  - Subsystem decomposition
  - Rigi overview window
- Generalization style
  - Class hierarchy
  - Inheritance hierarchy
- Layered style
  - Code in higher layers is only allowed to use code in lower layers
  - Operating system layers

11

What is a module?

- Software units with well defined interfaces providing a set of services
- 7 UML diagram types
- Module vs. component
  - Both are about decomposition
  - Module has a design time connotation and component a runtime connotation
- 4 common styles
  - The decomposition style – containment relationship among modules
  - The uses style – functional dependency relationships among modules
  - Generalization style – specialization relationships among modules
  - Layered style – allowed-to-use relation in a restricted fashion among modules

12

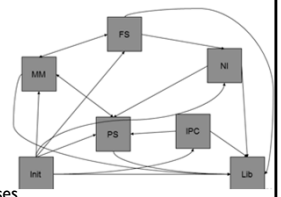
Purpose of module viewtype

- Construction
  - Blueprint for the source code
  - Modules and physical structures (source code files) will have close mapping
- Analysis
  - Requirements traceability
  - Impact analysis
- Communication
  - Useful for explaining the systems functionality
- Criteria for decomposition
  - Achievement of certain quality attributes—modifiability
  - Build-versus-buy decisions
  - Product line implementations

13

Component-and-connector viewtype

- Express runtime behaviour
- Styles
  - Pipe-and-filter style
  - Sockets
  - Shared-data style
  - Publish-subscribe style
  - Client-server style
  - 3-tier style
  - Peer-to-peer style
  - Communicating processes
  - Middleware is a connector
  - Repositories
  - 7 UML diagram types



14

Runtime models

- Elements having some runtime presence – processes, objects, clients, servers, data stores
- Pathways of interaction – communication links, protocols, information flows, access to shares storage

15

Purpose of component and connector viewtype

- To reason about runtime system quality attributes – performance, reliability, availability
- What are the systems principal executing components and how do they interact
- What are the major shared data resources
- Which parts of the system are replicated and how many times
- How does data progress through a system as it executes
- What protocols of interaction are used by communicating entities
- What parts of the system run in parallel
- How can the system's structure change as it executes

16

Allocation viewtype

- Maps software units to elements of the environment
  - Hardware, developers, managers, distributed teams
- Deployment style
- Implementation style
- Work assignment style

17

Siemens Views

- Applied Software Architecture
  - by Hofmeister, Nord and Soni, Siemens
- Conceptual view
- Execution view
- Module or subsystem view
- Code view

18

System Views

- Module or subsystem view
  - Often the number of modules in a system exceeds the number of lines per module
  - Partitioning work among programmers
  - Encapsulation of abstractions
  - Separated by interfaces
  - Layers of subsystems (levels of abstraction)
- Code view
  - Organization of source code into
    - object code, libraries, binaries
    - versions, files, directories, packages, modules, subsystems

19

System Views

- Conceptual view
  - Major design elements (entities) and relationships among them
  - Box and arrow diagrams during early design
- Execution view
  - Dynamic view
  - Communication, coordination, synchronization
  - Dynamic loading, I/O, scripting
  - Side effects, affecting devices

20

Conceptual View Applications

- How does the system fulfill the requirements?
- How are COTS components integrated and how do they interact with the rest of the system?
- How is domain-specific hardware and software integrated?
- How is functionality packaged into product releases?
- How are product lines supported?
- How can the impact of changes be minimized?

21

Module View Applications

- How is the product mapped to the software platform?
- What system/middleware support/services does it use?
- How is testing supported?
  - Testing harnesses, levels, regression test suites
- How are dependencies among subsystems minimized?
- How are changes insulated from COTS software?

22

Execution View Applications

- How does the system meet its performance, recovery, security, or reconfiguration requirements?
- How does the system balance resource usage (e.g., CPU load, memory)?
- How is concurrency, replication, or distribution achieved?
- How can the impact of changes be minimized on the run-time environment?

23

Code View Applications

- How can the time and effort for product efforts be reduced?
- How are versions and releases managed?
- How is the build time minimized?
- What tools are used for development?
- How are integration and testing supported?

24