

Software Architecture

Lecture 1 Introduction

João Pedro Sousa
George Mason University

outline

- what is software architecture?
 - relation to software engineering
 - relation to engineering
- architectural views
 - module, C&C, deployment
- the rest of the course

Acknowledgment

some of the material presented in this course is adapted from 17655,
taught to the MSE at CMU by David Garlan and Tony Lattanze

every system has an architecture

- just like every building has an architecture
 - and someone who acted as an architect



- an architecture may be
 - elegant and effective, or
 - clumsy and dysfunctional

examples of software architecture

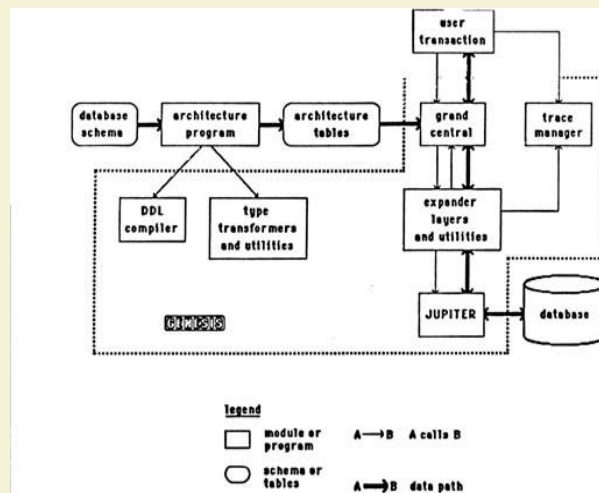


Figure 3.1 The Configuration of the GENESIS Prototype

Genesis: A Reconfiguration Database Management System, D. S. Batory, J.R. Barnett, J.F. Garza, K.F. Smith, K. Tsukuda, B.C. Twitchell, T.E. Wise, Department of Computer Sciences, University of Texas at Austin,

more examples

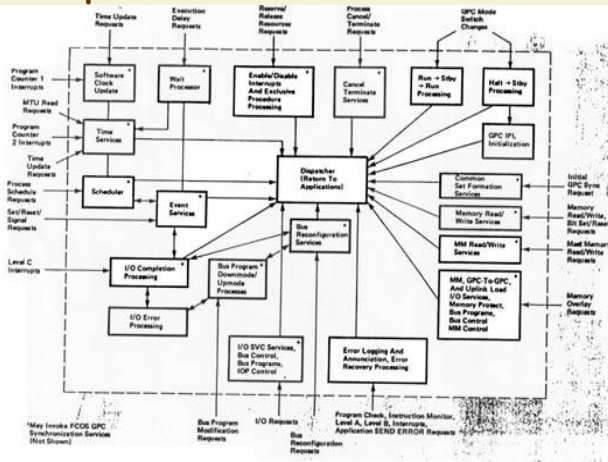


FIGURE 7. Flight Computer Operating System (The FCOS dispatcher coordinates and controls all work performed by the on-board computers.)

Communications of the ACM, "Architecture of the Space Shuttle Primary Avionics Software Systems," Gene D. Carlow, September 1984, Vol. 27, No. 9, P. 933

more examples

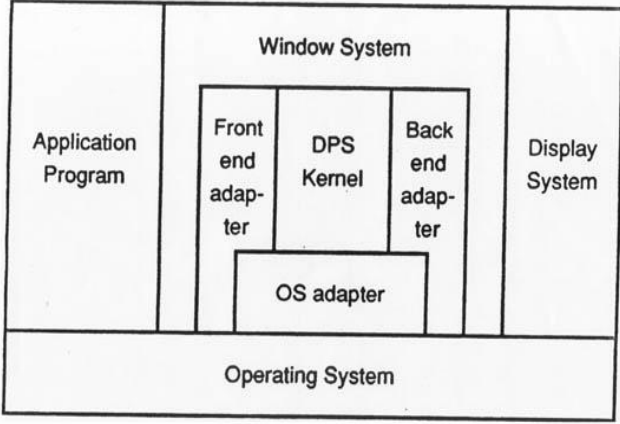
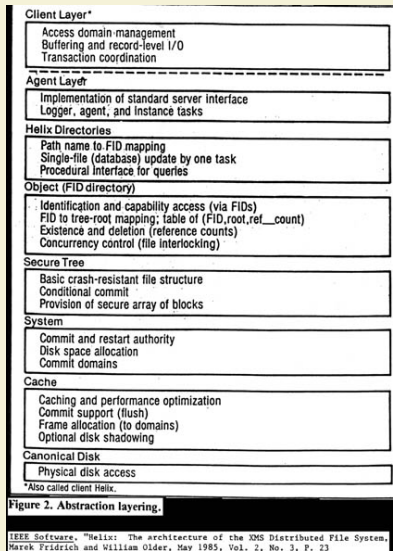


Figure 2. Display PostScript interpreter components.

An Overview of the DISPLAY POSTSCRIPT™ System, Adobe Systems Incorporated, March 16, 1988, P. 10

more examples



more examples

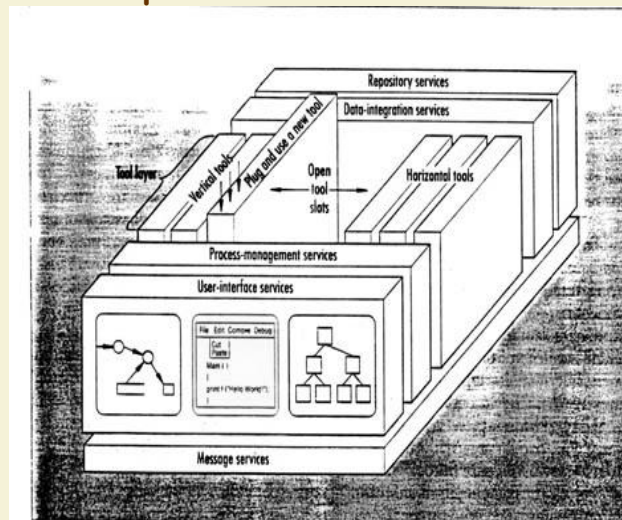
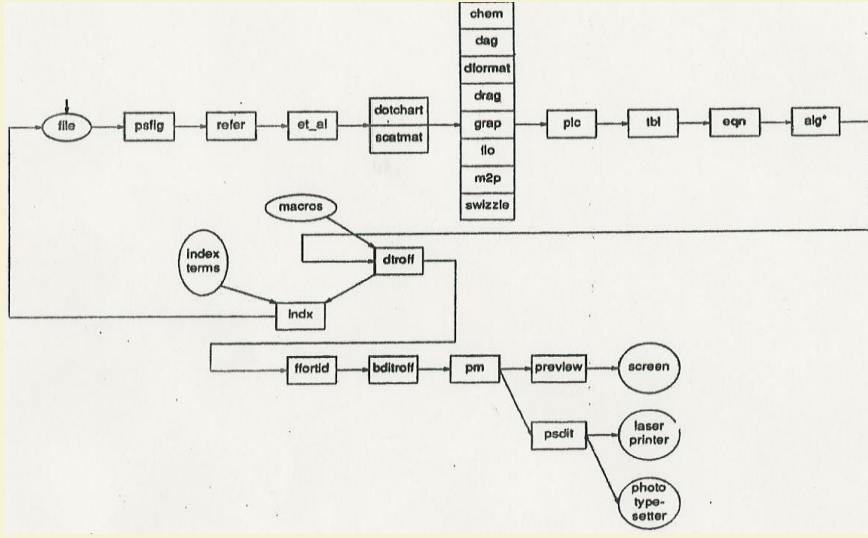


Figure 1. The NIST/ECMA reference model.

more examples



more examples

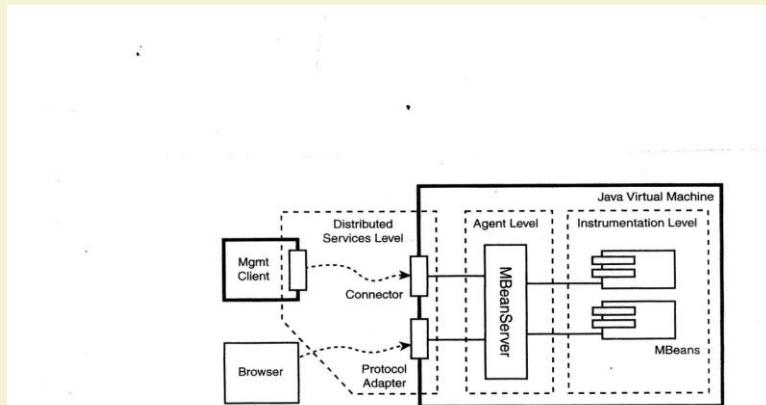
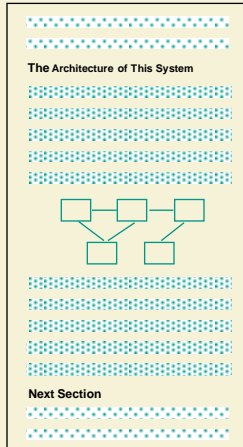


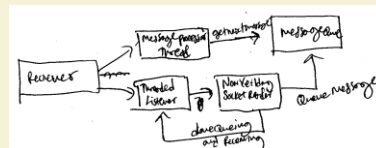
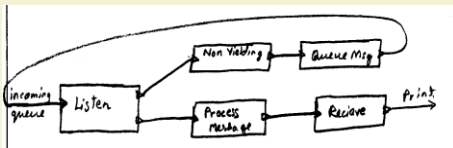
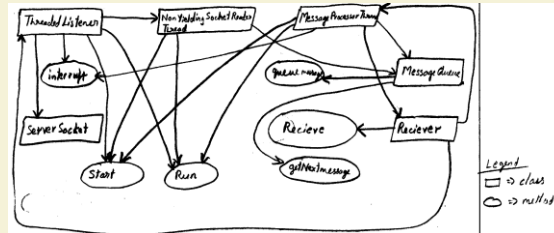
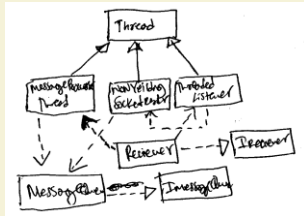
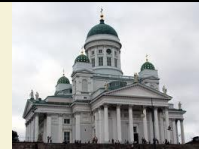
FIGURE 2.1
JMX Management Architecture.

description of a system's architecture



- usually informal prose plus box-and-line diagram
- lots of appeal to intuition
- little precision, rarely formal

unlike buildings software is hard to "see"



- architectures produced by different people looking at the same code

good architecture bestows:



- strength
 - durable
 - safe
 - beauty
 - balanced proportions
 - utility
 - fit planned uses
- reliable
 - fault-tolerant
 - ...
 - usable
 - intelligible
 - maintainable
 - ...
 - desired features
 - quality of service

[Vitruvius, ~40 BCE]

the challenge

- develop systems "architecturally"
 - apply codified architectural design expertise
 - build systems by assembling existing parts
 - use standards for integration
 - assure that the system has the desired properties
 - reduce development and evolution costs
- turn Software Architecture into an *engineering discipline*
 - from ad hoc definition to codified principles

what is software engineering?

quotes:

- software engineering is, in fact,
a form of engineering
David Parnas (modules, hiding, & OO)
- <it> is not, but it should be
Steve McConnell (Code Complete, Rapid Development)
- programming is an art and a science
Donald Knuth (computational complexity, analysis of algorithms)

what is engineering?

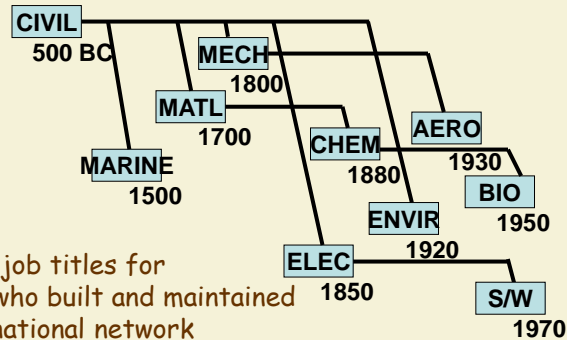
definitions abound. They have in common:

Creating cost-effective solutions ...
 ... to practical problems ...
 ... by applying scientific knowledge ...
 ... building things ...
 ... in the service of mankind

engineering enables ordinary people
 to do things that formerly required virtuosos
 engineering entails making decisions under the
 constraints of limited time, knowledge, and resources

what is engineering?

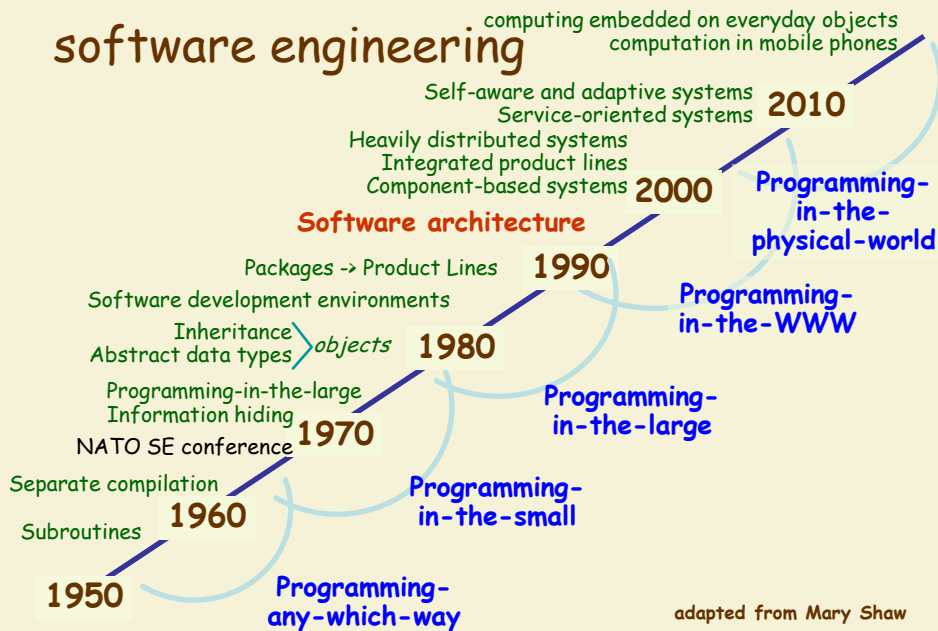
- before the 19th century the kinds of *systems* that engineers studied and built had little technological or mechanical aspects



- by 1920, AT&T had job titles for *System Engineers*, who built and maintained *The System*: their national network

Honour, E. "Characteristics of Engineering Disciplines"
13th Int'l Conference on Systems Engineering, 1999

software engineering



many angles into software engineering

- managing software production
 - who does what, when - lifecycle/process/risk management
 - methods, tools, skills
- best practices/tools of the trade
 - systematic testing/peer reviews
 - separation of concerns/aspects
 - rapid prototyping/configuration management/evolution
 - product lines/design patterns
- correlating structure and properties
 - software economics: size/complexity \leftrightarrow cost
 - **Software Architecture:**
design patterns \leftrightarrow *-ility tradeoffs

software architect: a recognized trade within SE



human-computer interaction

applications

development/execution envs.

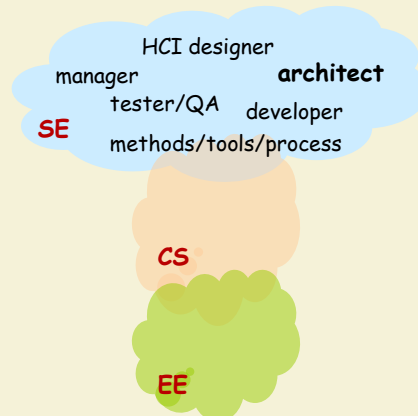
middleware

OS/storage management

networking

hardware

interfaces with:
psychology, ergonomics, human factors



what is software architecture?

Definitions abound. They have in common:

the structure

or structures of the system
which comprise elements
their externally-visible properties
and the relationships among them

SA & programming address different issues

Architecture

interactions among parts

system-wide views

declarative

more stable components

system-level performance

- software engineering is, in fact, a form of engineering
David Parnas

Programs

implementations of parts

local views

operational

dynamic allocation

algorithmic performance

- programming is an art and a science
Donald Knuth

SA makes a difference

- design Reviews at ATT
 - Architecture Review Board created 1988
 - facilitates architectural reviews for projects
- results
 - reviewed over 350 projects from 1989-95
 - "a correct architecture has the largest single impact on cost and quality of the product" (Maranzano 1995)
 - "based on experience ... we found an average savings of at least one-half staff year per project reviewed, and substantially larger savings on several projects reviewed."
 - estimated cumulative savings: approx 175 staff years

SA in the 2000's

- incorporation of architectural notions into mainstream *design languages* (e.g., UML-2), and *tools* (e.g., Rose)
- *methods* based on architectural design and refinement (e.g., Model-Driven Architecture - MDA)
- some architecture *analysis tools*
- architectural *standards* for enterprise systems (e.g., RM-ODP, TOGAF)

AcmeStudio - TempControlSystem.acme - AcmeStudio

File Edit View Navigate Search Project Run Design Family Window Help

Navigator: RainbowMDS, docs, families, images, Mds, tests, Rocky7.acme, TempControlSystem.acme, Topology.acme

Outline: anMDSL, EXEC, TCON, CTSV, TEST, SHSV, SACT, TSEN, conn01, stUpdConn1, stNotfConn2, stUpdConn2, conn03, cmdNotfConn1, conn0, cmdSubConn1, stNotfConn1, conn05

Global Typespace: Component, MDSFam, ActuatorT, ControllerT, EstimatorT, ExecutableT, ExecutableT, HealthStateVarT, SchedulerT, SensorT, StateVarT, CommandNotfConnT, CommandQueryConnT, CommandSubmitConnT, ConstraintExecConnT, ExecuteConnT, MeasurementNotfConnT, MeasurementQueryConnT, PointToPointConnT, PubSubConnT

Acme Source Family - MDSFam System - anMDSL

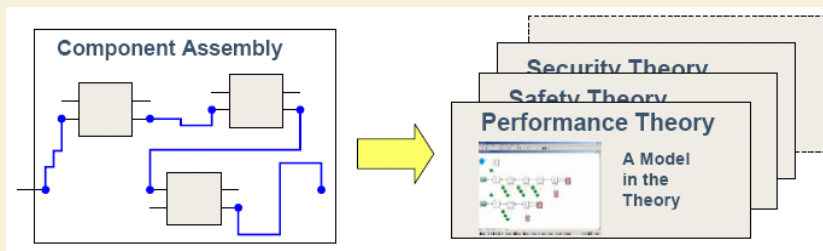
Element View: Name: TEST, Description: Properties, Rules, Structure, Representations, Errors, Types

Properties	Rules	Structure	Representations	Errors	Types
▲	Rule 2.2: An estimator that is not event driven can have no command notification ports				
▲	Rule 3.1: An estimator that needs no command evidence can have no command query ports				
▲	Rule 4.1: An estimator that requires no measurement data can have no measurement query ports				
▲	Rule 5.2: An estimator that is not event driven can have no measurement notification ports				
▲	Rule 7.2: Each estimator state update port must be connected to only one state variable.				
▲	Estimators can only have ports of type StateUpdateReqPortT, StateQueryReqPortT, CommandQueryReqPortT, Co...				
▲	Invariant Forall p in self.ports □ Exists t in {commandNotfProvPortT, CommandQueryReqPortT, MeasurementQu...				

Element View Tasks

SWE 443 - Software Architecture © Sousa 2011 Lecture 1 - Intro - 25

SA correlates design patterns/styles with quality attribute tradeoffs



- using formal/quantitative models of systems and environment

focus on **run-time** structure & **quality attributes**

- performance
- security
- availability
- recoverability
- safety
- interoperability
- portability
- scalability
- usability
 - learnability
 - fit user goals
 - customizability
 - responsiveness
 - ...
- maintainability
 - extensibility
 - testability
 - buildability
 - and more...

take 5

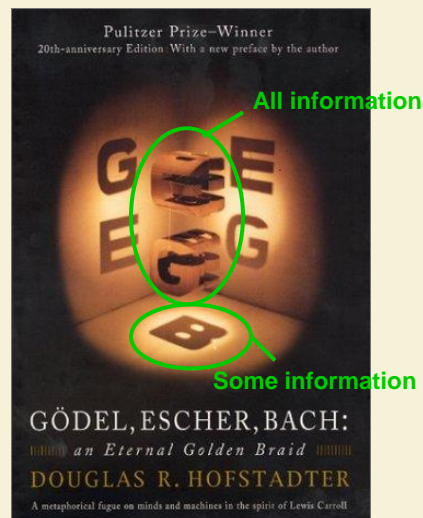
outline

- what is software architecture?
- architectural views
 - module viewtype
 - component & connector viewtype
 - allocation viewtype
 - styles
- the rest of the course

one system, many views

- a **view** is a representation of a set of system elements and the relations among them
- not *all* system elements
- a view selects *element types* and *relation types* of interest, and shows only those

why?



one system, many views

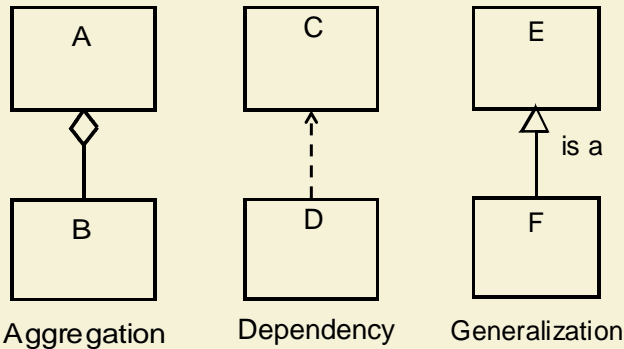
- an architect examines the system in three ways
 - how is it structured as a set of **code units**?
 - **module** viewtype
 - how is it structured as a set of elements that have **run-time behavior and interactions**?
 - **component & connector** viewtype
 - how does it relate to non-software structures, such as **hardware** and development **teams**?
 - **allocation** viewtype

module viewtype describes the code structure

- **elements** are **modules**
code unit that implements a set of functionalities
- **relations** among modules include
 - **A is part of B**
defines a part-whole relation
 - **A depends on B**
defines a functional dependency relation
 - **A is a B**
defines specialization and generalization

different notations exist for module views

- UML class diagrams:



- informal: stacked boxes, box-and-line...
examples in a moment

module viewtype used for code construction and budgeting

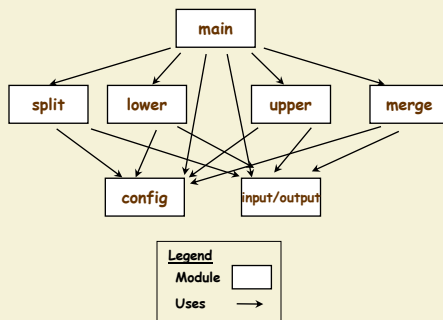
- construction
 - module views are the blueprints for the code
 - modules are assigned to teams for implementation
 - modules are often the unit for refining the design (e.g., module interfaces)
- analysis
 - traceability and impact analysis
 - budgeting, project management: planning and tracking

module and C&C show different aspects

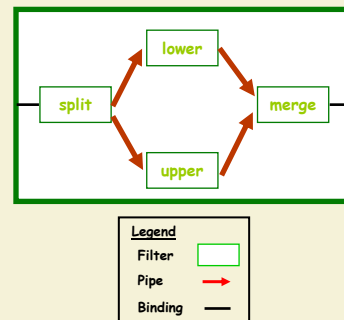
example program:

- produce alternating case of characters in a stream

module view



C&C view

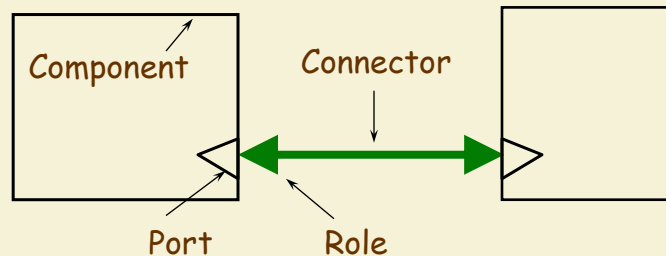


C&C viewpoint describes how the system works

- **elements**
 - **components** (boxes)
principal units of run-time computation and data stores
 - **connectors** (lines)
interaction mechanisms - identity and behavior of their own
- **relations**
 - **attachment** of components to connectors
- **properties**
information for construction & analysis
 - quality attributes
 - others, depending on *style* (more in a moment)

different notations exist for C&C views

- ACME diagrams:



- other notations (normally box-and-line)

examples in a moment

C&C viewtype used for behavior and QoS analysis

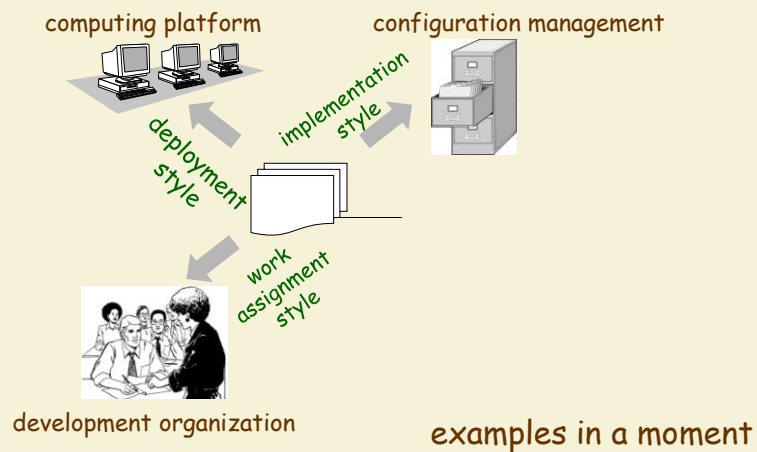
- construction
 - how the system will appear at run time
 - what kind of behavior must be built in
 - pathways of interaction and communication mechanisms
- analysis of runtime properties
 - availability
 - performance
 - security
 - reliability...

allocation viewtype

- **elements**
 - **software** elements
as defined in module or C&C views
 - **environment** elements
such as **hardware** and development **teams**
- **relations**
 - **allocated-to**

notations for allocation views depend on the *style*

- normally informal, style-specific notations



outline

- what is software architecture?
- architectural views
 - overview of the first half semester
 - module viewtype
 - component & connector viewtype
 - allocation viewtype
 - styles
- the rest of the course

architectural styles: specialization of element and relation types

- within each viewtype, recurring forms have been widely observed in different systems
- these forms are worth capturing because they have known properties and can be re-used:
"tools" in the architect's "bag of tricks"

an architectural style
is a specialization of element and relation types
together with a set of constraints on how they can be used

- styles exist independently of any system
- two different systems can use the same style
- different parts of the same system may use different styles

remember

- *viewtypes* reflect the three broad ways an architect looks at a system:
 - units of implementation (*module* viewtype)
 - run-time units (*C&C* viewtype)
 - relation to non-software structures (*allocation* viewtype)
- within a viewtype, many choices remain:
 - what kinds of elements are allowed
 - how they relate to each other
 - how are they used or configured
 different *styles* result from making different choices

three major styles in the module viewtype

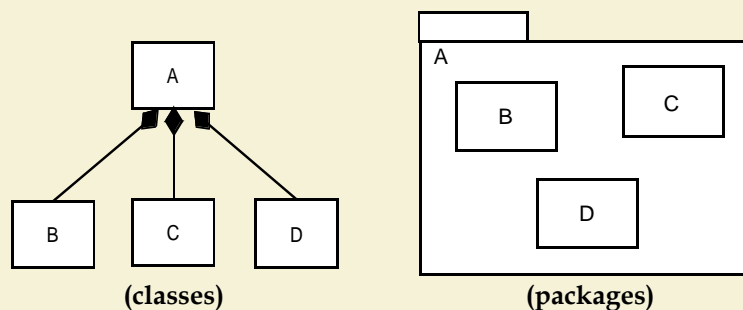
- *decomposition* style
 - hierarchical decomposition of modules
 - supports concurrent development
- *generalization* style
 - specialization hierarchy
 - supports reuse; managing large numbers of definitions
- *layered* style
 - virtual machines
 - supports portability, reuse

decomposition style in the module viewpoint

- elements are modules
- relations restricted to A is part of B
- what it is for
 - a starting point
frequently, assigning functions to modules
is a prelude to detailed design
 - change/impact analysis
 - basis for work assignments
provides elements in the allocation view

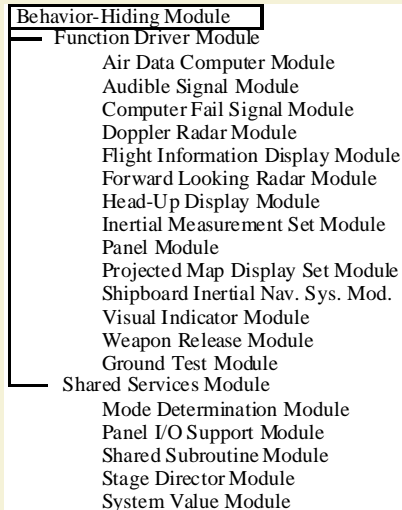
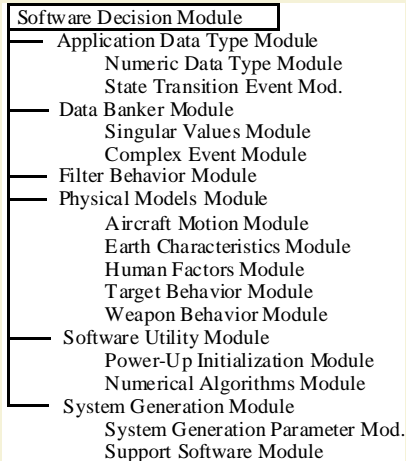
decomposition style in the module viewpoint

- examples in UML



decomposition style in the module viewpoint

- outline/tree examples



SWE 443 - Software Architecture

© Sousa 2011

Lecture 1 - Intro - 47

generalization style in the module viewpoint

- elements are modules
- relations restricted to *A is a B*
- properties
 - inheritance semantics: interface vs. implementation
- what it is for
 - basis for object-oriented designs
 - supports evolution and extension
 - reuse

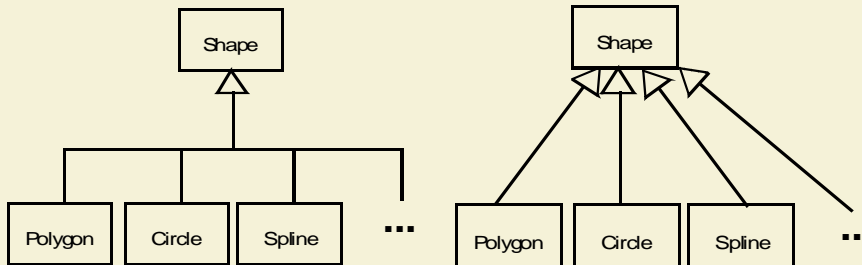
SWE 443 - Software Architecture

© Sousa 2011

Lecture 1 - Intro - 48

generalization style in the module viewpoint

- examples in UML



- reflected in programming languages
 - Circle **extends** Shape

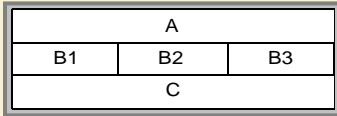
layered style in the module viewpoint

- **elements** are **layer** modules
- **relations** restricted to **A allowed to use B**
a special case of **A depends on B**
- **stylistic rules**
 - every piece of software is assigned to exactly one layer
 - software in a layer is allowed to use software in {any lower layer, next lower layer}
 - software in a layer {is, is not} allowed to use other software in same layer
- **what it is for**
 - separation of concerns/incremental development
 - portability

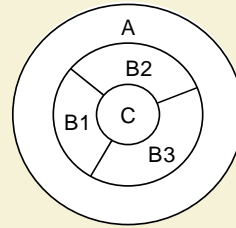
style variations:

layered style in the module viewpoint

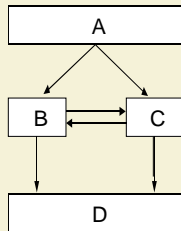
- examples (interpret each one)
- stack of boxes



- concentric rings

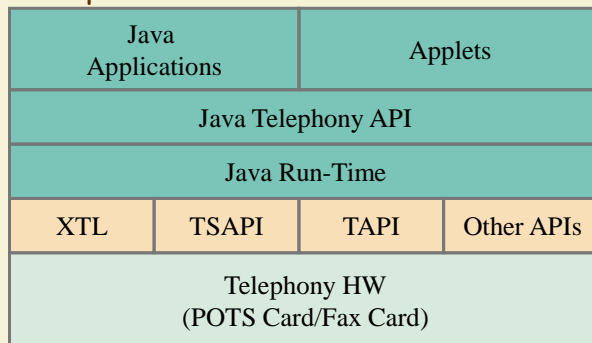


- boxes and arrows



layered style in the module viewpoint

- example: JavaPhone



(JTAPI page 7)

- is this a good description?
(interpret it according to the style variations)

many styles in the C&C viewtype

data flow

batch sequential
dataflow network (pipe & filter)
acyclic, fan-out, pipeline, Unix
closed loop control

call-and-return

main program/subroutines
information hiding
objects, naive client-server

interacting processes

communicating processes
LW processes, distributed objects
event systems
implicit invocation
publish-subscribe

data-oriented repository

transactional databases
true client-server
blackboard
modern compiler

data-sharing

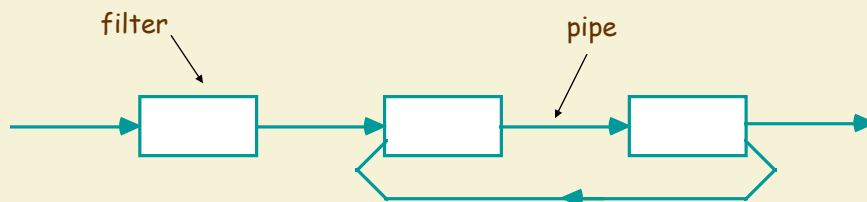
compound documents
hypertext
Fortran COMMON
LW processes

hierarchical

tiers
interpreter
N-tiered client-server

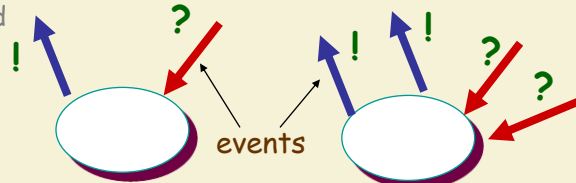
pipe & filter style in the C&C viewtype

- **elements** are **pipes** (data flow) and **filters** (computation)
- **relations** restricted to P.in/out **attached to F.port**
- **what it is for**
 - functionality related to data streaming and transformation
e.g. media streaming, image processing...



event publish-subscribe style in the C&C viewtype

- **elements** are **objects/threads** and **events**
- **relations** restricted to A **publishes** E, A **subscribes** E
- two style variants
 - implicit invocation: one responder will be passed the event
 - publish-subscribe: zero or many subscribers (no guaranties)
- what it is for
 - high degree of separation between functional units
e.g. Google Android



SWE 443 - Software Architecture

© Sousa 2011

Lecture 1 - Intro - 55

three major styles in the allocation viewtype

- **deployment** style
 - allocates software elements, i.e. code, to processing and communication nodes
 - properties include those necessary to calculate (and achieve) performance, availability
- **implementation** style
 - allocates software elements to structures in the development environment's file systems
 - properties include files and capacities
- **work assignment** style
 - allocates software elements to organizational work units
 - properties include skill sets

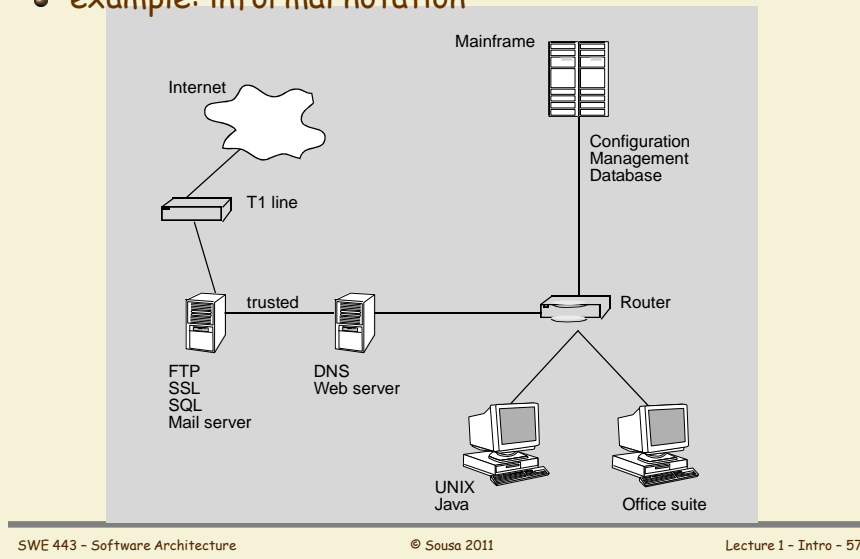
SWE 443 - Software Architecture

© Sousa 2011

Lecture 1 - Intro - 56

deployment style in the allocation viewtype

- example: informal notation



in Summary

- **views** help manage the complexity of describing an architecture
- **viewtypes** determine the kinds of things a view talks about
 - three primary viewtypes: **module**, **C&C**, **allocation**
- each viewtype has many **styles**
 - **module**: decomposition, generalization, layered, ...
 - **C&C**: pipe & filter, client-server, pub-sub...
 - **allocation**: deployment, work assignment...

why study SA?

<http://money.cnn.com/magazines/moneymag/bestjobs/2010/>

CNN Money.com
A Service of CNN, Fortune & Money

Home Business News Markets Personal Finance Retirement Technology Small Business Fortune Video My Preferences CNN.com

BEST JOBS IN AMERICA MoneyPayscale.com's list of great careers | 2010

Full List High Pay Job Growth Quality of Life Sectors

1. Software Architect Recommend 9K 1 of 100 Next

Top 100 rank: 1
Sector: Information Technology

What they do: Like architects who design buildings, they create the blueprints for software engineers to follow – and pitch in with programming too. Plus, architects are often called on to work with customers and product managers, and they serve as a link between a company's tech and business staffs.

What's to like: The job is creatively challenging, and engineers with good people skills are liberated from their screens. Salaries are generally higher than for programmers, and a typical day has more variety.

"Some days I'll focus on product strategy, and other days I'll be coding down in the guts of the system," says David Chaiken, 46, of Yahoo in Sunnyvale, Calif., whose current projects include helping the web giant customize content for its 600 million users. Even though programming jobs are moving overseas, the face-to-face aspect of this position helps cement local demand.

What's not to like: You are often outside the management chain of command, making it hard to get things done.

Requirements: Bachelor's degree, and either a master's or considerable work experience to demonstrate your ability to design software and work collaboratively.

Software Architect job openings jobs by simplyhired

Web Software Architect
New York, NY - Manual Enrollment

LEAD SOFTWARE ARCHITECT
Irving, TX - Verizon Communications

Lead Developer and Software Architect
Golden, CO - Ingersoll Rand

Senior Software Architect
Warren, PA - Harsco

See All Jobs

Search All Jobs from Across the Web

job title or company location

How would you use our technology to make a better world?

Learn how to share your ideas at toyota.com/ideasforgood

ideas for good **TOYOTA** **GET STARTED**

Sponsored Links

Today's Stock Pick: HRTX
Growing Wind Energy Developer Revenue-Producing

SWE 443 - Software Architecture

© Sousa 2011

Lecture 1 - Intro - 59

outline

- what is software architecture?
- architectural views
- the rest of the course

SWE 443 - Software Architecture

© Sousa 2011

Lecture 1 - Intro - 60