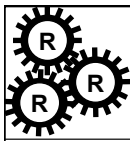


System Design, Part II

Improving Designs

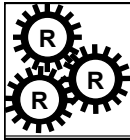
CSE870: Advanced Software Engineering (System Design) Cheng



5.5 Achieving Quality Attributes Performance

- Performance attributes describe constraints on system speed and capacity:
 - Response time: How fast does our software respond to requests?
 - Throughput: How many requests can it process per minute?
 - Load: How many users can it support before response time and throughput start to suffer?

CSE870: Advanced Software Engineering (System Design) Cheng

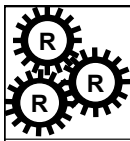


5.5 Achieving Quality Attributes

Performance

- Tactics for improving performance include:
 - Improve utilization of resources
 - Manage resource allocation more effectively
 - First-come/first-served: Requests are processed in the order in which they are received
 - Explicit priority: Requests are processed in order of their assigned priorities
 - Earliest deadline first: Requests are processed in order of their impending deadlines
 - Reduce demand for resources

CSE870: Advanced Software Engineering (System Design) Cheng

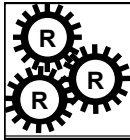


5.5 Achieving Quality Attributes

Security

- Two key architectural characteristics particularly relevant to security: immunity and resilience
- **Immunity**: ability to thwart an attempted attack
 - The architecture encourages immunity by:
 - Ensuring all security features are included in the design
 - Minimizing exploitable security weaknesses
- **Resilience**: ability to recover quickly and easily from an attack
 - The architecture encourages resilience by:
 - Segmenting functionality to contain attack
 - Enabling the system to quickly restore functionality

CSE870: Advanced Software Engineering (System Design) Cheng

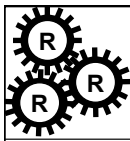


5.5 Achieving Quality Attributes

Reliability

- A software system is reliable if it correctly performs its required functions under assumed conditions
 - Is the software internally free of errors?
- A **fault** is the result of human error, compared to a **failure**, which is an observable departure from required behavior
 - Software is made more reliable by preventing or tolerating faults

CSE870: Advanced Software Engineering (System Design) Cheng

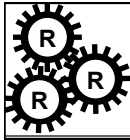


5.5 Achieving Quality Attributes

Reliability (continued)

- **Passive fault detection:** wait until fault occurs during execution
- **Active fault detection:** periodically check for symptoms or try to anticipate when failures will occur
- **Exceptions:** situations that cause the system to deviate from its desired behavior
- Include **exception handling** in design to handle exception and return system to acceptable state
- Typical exceptions include:
 - Failing to provide a service
 - Providing the wrong service
 - Corrupting data
 - Violating a system invariant (e.g.; security property)
 - Deadlocking

CSE870: Advanced Software Engineering (System Design) Cheng

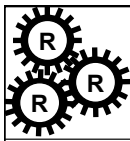


5.5 Achieving Quality Attributes

Reliability (continued)

- N-version programming
 - If two functionally equivalent systems are designed by two different design teams at two different times using different techniques, the chance of the same fault occurring in both implementations is very small
 - N-version programming has been shown to be less reliable than originally thought, because many designers learn to design in similar ways, using similar design patterns and principles

CSE870: Advanced Software Engineering (System Design) Cheng

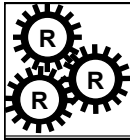


5.5 Achieving Quality Attributes

Reliability (continued)

- **Fault recovery:** handling fault immediately to limit damage
- **Fault recovery tactics:**
 - Undoing transactions: manage a series of actions as a single transaction that are easily undone if a fault occurs midway through the transaction
 - Checkpoint/rollback: software records a checkpoint of current state; rolls back to that point if system gets in trouble
 - Backup: system automatically substitutes faulty unit with backup
 - Degraded service: returns to previous state, offers degraded version of the service
 - Correct and continue: detects the problem and treats the symptoms
 - Report: system returns to its previous state and reports the problem to an exception-handling unit

CSE870: Advanced Software Engineering (System Design) Cheng

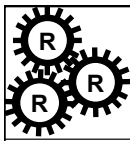


5.5 Achieving Quality Attributes

Sidebar 5.6 The Need for Safe Design

- From 1986 to 1997, over 450 reports filed with the U.S. Food and Drug Administration (FDA) detailing software defects in medical devices, 24 of which led to death or injury
 - Numbers may be greater based on time to file report
- The FDA established a software forensics unit in 2004 after noticing that medical device makers were reporting more and more software-based recalls
- Software designers must see directly how their products will be used
- Then designers can build in preventative measures to ensure their products are not misused

CSE870: Advanced Software Engineering (System Design) Cheng

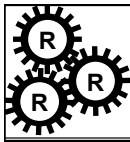


5.5 Achieving Quality Attributes

Robustness

- A system is **robust** if it includes mechanisms for accommodating or recovering from problems in the environment or in other unit
- Mutual suspicion: each software unit assumes that the other units contain faults
- Robustness tactics differ from reliability tactics
- Recovery tactics are similar:
 - Rollback to checkpoint state
 - Abort a transaction
 - Initiate a backup unit
 - Provide reduced service
 - Correct symptoms and continue processing
 - Trigger an exception

CSE870: Advanced Software Engineering (System Design) Cheng

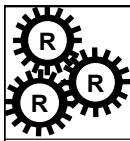


5.5 Achieving Quality Attributes

Usability

- Usability reflects the ease in which a user is able to operate the system
 - User interface should reside in its own software unit
 - Some user-initiated commands require architectural support
 - There are some system-initiated activities for which the system should maintain a model of its environment

CSE870: Advanced Software Engineering (System Design) Cheng

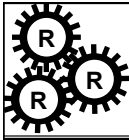


5.5 Achieving Quality Attributes

Business Goals

- Business Goals are quality attributes the system is expected to exhibit (e.g., minimizing the cost of development and time to market)
 - Buy vs. Build
 - Save development time, money
 - More reliable
 - Existing components create constraints; vulnerable to supplier
 - Initial development vs. maintenance costs
 - Save money by making system modifiable
 - Increased complexity may delay release; lose market to competitors
 - New vs. known technologies
 - Acquiring expertise costs money, delays product release
 - Either learn how to use the new technology or hire new personnel
 - Eventually, we must develop the expertise ourselves

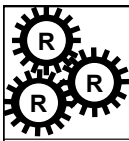
CSE870: Advanced Software Engineering (System Design) Cheng



5.6 Collaborative Design

- Usually the design of software systems is performed by a team of developers
- Several issues must be addressed by the team:
 - Who is best suited to design each aspect of the system
 - How to document all aspects
 - How to coordinate and integrate the software units
- Important to view group interaction in its cultural and ethical contexts

CSE870: Advanced Software Engineering (System Design) Cheng

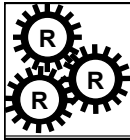


5.6 Collaborative Design

Sidebar 5.7 The Causes of Design Breakdown

- Each team member must be aware of the causes of design breakdowns and use the team's strengths to address them
- The main types of process breakdown are:
 - Lack of specialized data schemas
 - Lack of meta-schema about the design process
 - Poor prioritization of issues
 - Difficulty in considering constraints
 - Difficulty in performing mental simulations
 - Difficulty in tracking and returning to subproblems
 - Difficulty in expanding or merging solutions

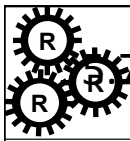
CSE870: Advanced Software Engineering (System Design) Cheng



5.6 Collaborative Design Outsourcing

- Coordination becomes increasing difficult
- Collaborative team may be distributed around the world
- Four stages in distributed development:
 - Project performed at single site with on-site developers from foreign countries
 - On-site analysts determine system requirements, which are in turn provided to off-site groups
 - Off-site developers build generic products and components that are used worldwide
 - Off-site developers build products that take advantage of their individual areas of expertise

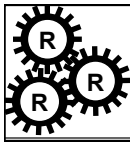
CSE870: Advanced Software Engineering (System Design) Cheng



7 Architecture Evaluation and Refinement

- Design is iterative: we propose design decisions, assess, make adjustments, and propose more decisions
- Many techniques to evaluate the design:
 - Measuring design quality
 - Safety analysis
 - Security analysis
 - Trade-off analysis
 - Cost-benefit analysis
 - Prototyping

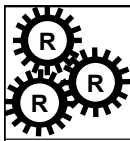
CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Measuring Design Quality

- Metrics being developed to access key aspects of design quality
 - Chidamber and Kemerer
 - General set of metrics applicable to object-oriented systems
 - Briand, Morasca, and Basili
 - Metrics for evaluating high-level design, including cohesion and coupling
 - Briand, Devanbu, and Melo
 - Build on above ideas to propose ways to measure coupling

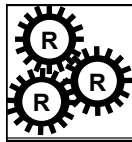
CSE870: Advanced Software Engineering (System Design) Cheng



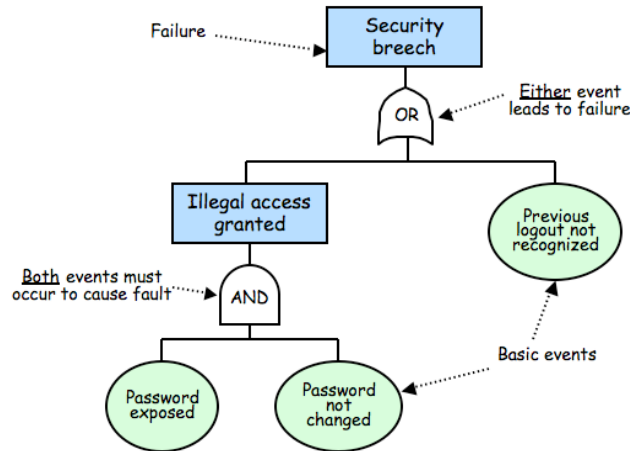
5.7 Architecture Evaluation and Refinement Safety Analysis

- Several techniques during design to identify possible faults
- **Fault-tree analysis** traces backwards through a design
 - Trees then used to determine which faults to correct/avoid/tolerate
 - **Data-flow graph**: depicts the transfer of data from one process to another
 - **Control-flow graph**: depicts possible transfer of control among software units

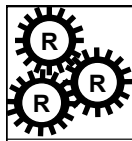
CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Safety Analysis (continued)



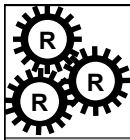
CSE870: Advanced Software Engineering (System Design) Cheng



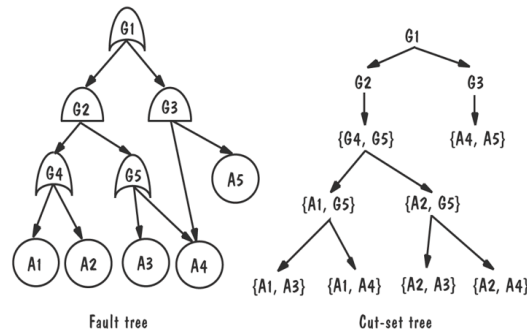
5.7 Architecture Evaluation and Refinement Safety Analysis (continued)

- Once fault tree is constructed we search for weaknesses
- Cut-set tree reveals event combinations can cause failure
 - Rules for forming cut-set tree:
 - Assign the top node of the cut-set tree to match the logic gate at the top of the fault tree.
 - Working from the top down, expand the cut-set tree as follows:
 - Expand an *or*-gate node to have two children, one for each *or*-gate child
 - Expand an *and*-gate node to have a child composition node listing both of the *and*-gate children
 - Expand a composition node by propagating the node to its children, but expanding one of the gates listed in the node
 - Continue until all leaf nodes are basic events or composition nodes of basic events

CSE870: Advanced Software Engineering (System Design) Cheng

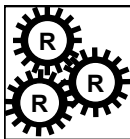


5.7 Architecture Evaluation and Refinement Safety Analysis (continued)



- Once fault is found in design:
 - Correct the fault
 - Add components or conditions to prevent
 - Add components that detect fault and recover from damage


CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Security Analysis

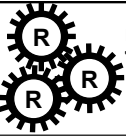
- Six steps to performing security analysis:
 - Software characterization: [review documentation for understanding functionality of the system](#)
 - Threat analysis: look for threats (e.g., espionage, interception, disruption)
 - Vulnerability assessment: includes failure to authenticate user or use of cryptological algorithm that is easy to break
 - Risk likelihood determination: must consider motivation, ability of the threat to exploit, impact of the exploitation, and degree to which current controls can prevent
 - Risk impact determination: business consequences
 - Risk mitigation planning: planning to reduce likelihood and consequences of most severe risks

CSE870: Advanced Software Engineering (System Design) Cheng



STOP

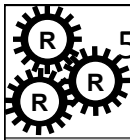
CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Trade-off Analysis

- Often several alternative designs to consider
 - professional duty to explore design alternatives and not simply implement the first design that comes to mind
 - different members of design team may promote competing designs
 - need a measurement-based method for comparing design alternatives

CSE870: Advanced Software Engineering (System Design) Cheng

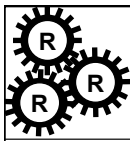


5.7 Architecture Evaluation and Refinement

One Specification, Many Designs

- **One specification, many designs:** to see how different designs can be used to solve the same problem
- Shaw and Garlan present four different architectural designs to implement KWIC (Key Word in Context problem)
 - shared data
 - abstract data type
 - implicit invocation
 - pipe and filter

CSE870: Advanced Software Engineering (System Design) Cheng



KWIC Example

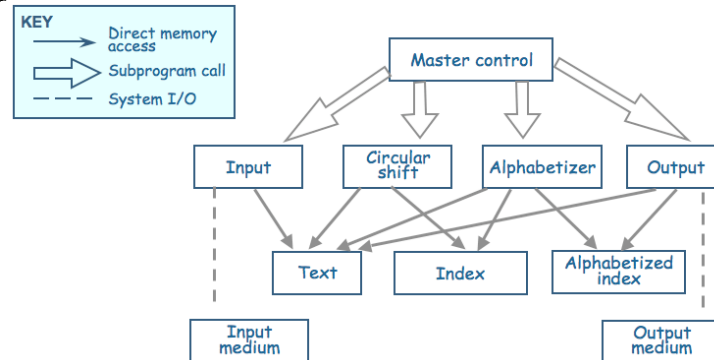
- **Inputs: *Sequence of lines***
Pipes and Filters
Architectures for Software Systems
- **Outputs: *Sequence of lines, circularly shifted and alphabetized***
and Filters Pipes
Architectures for Software Systems
Filters Pipes and
for Software Systems Architectures
Pipes and Filters
Software Systems Architectures for
Systems Architectures for Software

CSE870: Advanced Software Engineering (System Design) Cheng

5.7 Architecture Evaluation and Refinement

One Specification, Many Designs (continued)

- **Shared data solution**
- Four functional parts: input, circular shift, alphabetize, and output

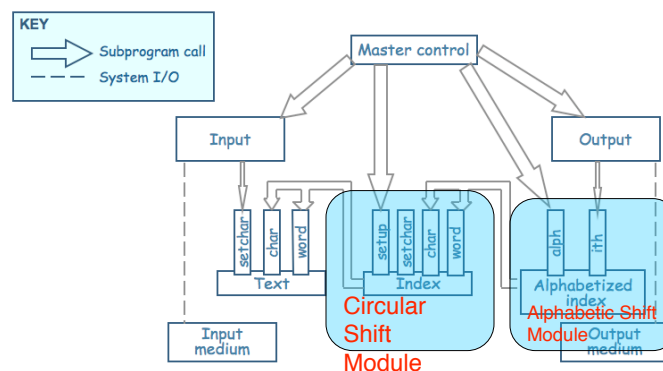


CSE870: Advanced Software Engineering (System Design) Cheng

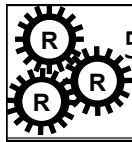
5.7 Architecture Evaluation and Refinement

One Specification, Many Designs (continued)

- **Data-module solution**
- Modules form data abstraction (hide data representation)



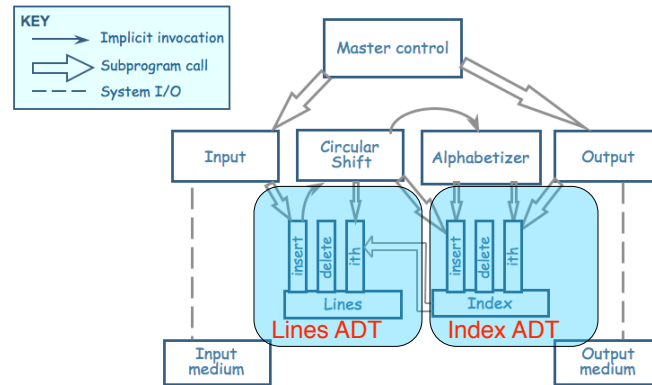
CSE870: Advanced Software Engineering (System Design) Cheng



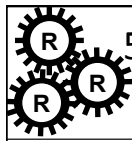
5.7 Architecture Evaluation and Refinement

One Specification, Many Designs (continued)

- **ADT solution:** Data are no longer centralized, stored, and shared, but the decomposition process is similar



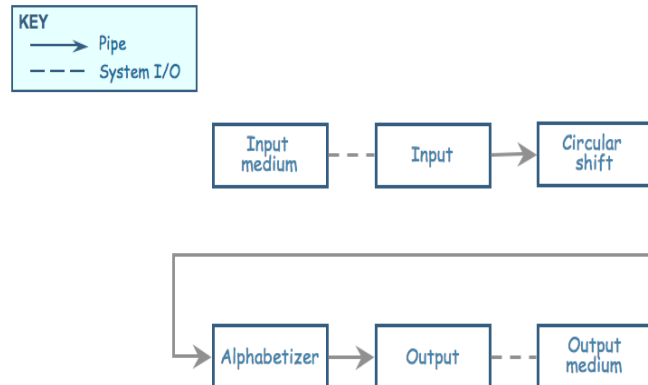
CSE870: Advanced Software Engineering (System Design) Cheng



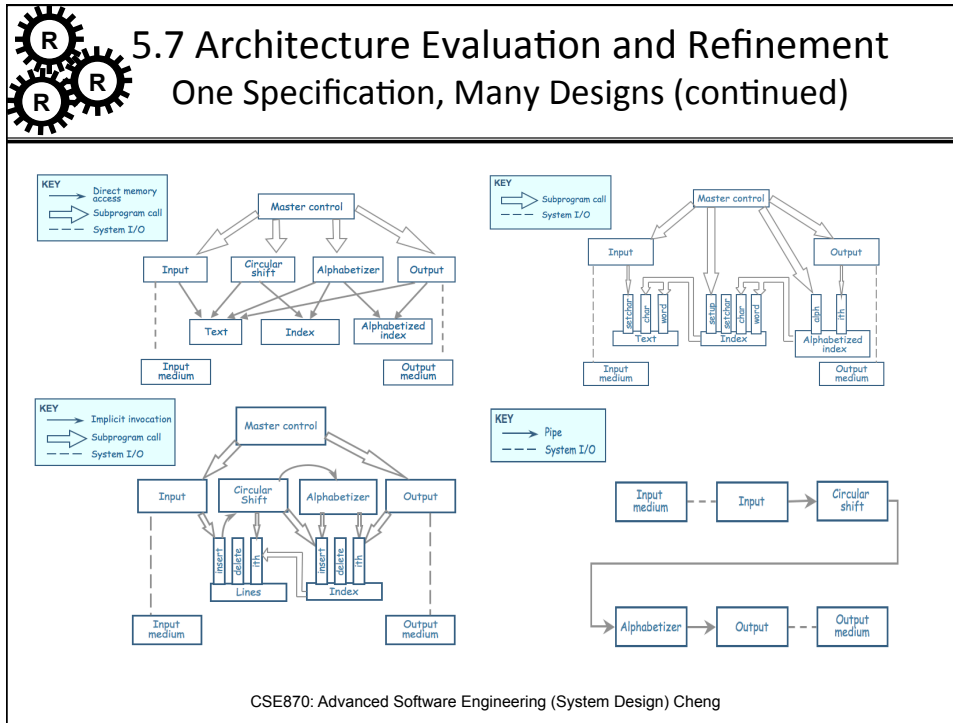
5.7 Architecture Evaluation and Refinement

One Specification, Many Designs (continued)

- **Pipe-and-filter solution:** The sequence of processing is controlled by the sequence of filters



CSE870: Advanced Software Engineering (System Design) Cheng

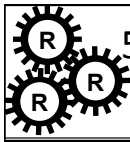


5.7 Architecture Evaluation and Refinement One Specification, Many Designs (continued)

- (Coarse-grained) Comparison of KWIC solutions

Attribute	Shared Data	Data Abstraction	Implicit Invocation	Pipe and Filter
Easy to change Algorithm	-	-	+	+
Easy to Change Data	-	+	-	-
Easy to Add Functionality	+	-	+	+
Performance	-	-	+	+
Efficient Data Rep	+	+	+	-
Easy to Reuse	-	+	-	+

CSE870: Advanced Software Engineering (System Design) Cheng



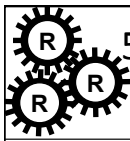
5.7 Architecture Evaluation and Refinement

One Specification, Many Designs (continued)

- Weighted comparison of KWIC solutions

Attribute	Priority	Shared data	Abstract data type	Implicit invocation	Pipe and filter
Easy to change algorithm	1	1	2	4	5
Easy to change data representation	4	1	5	2	1
Easy to change function	3	4	1	4	5
Good performance	3	5	4	2	2
Easy to reuse	5	1	4	2	5
Total		36	57	40	55

CSE870: Advanced Software Engineering (System Design) Cheng

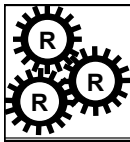


5.7 Architecture Evaluation and Refinement

One Specification, Many Designs (continued)

- Other attributes to consider
 - Modularity
 - Testability
 - Security
 - Ease of use
 - Ease of understanding
 - Ease of integration

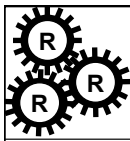
CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Cost-Benefit Analysis

- Consider a proposal to improve KWIC performance because the number of KWIC indices have increased
 - Eliminate noise word indices?
 - Change representation of indices to bin of indices?
 - Increase server capacity?
- A cost–benefit analysis is a widely used business tool for estimating and comparing the costs and benefits of a proposed change

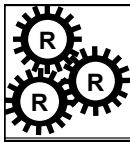
CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Computing Benefits

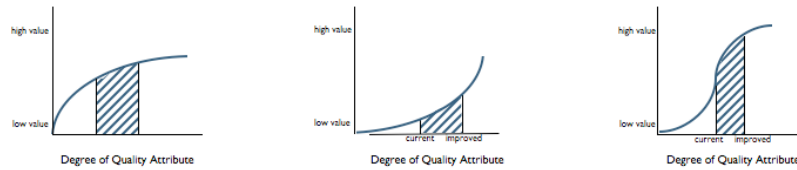
- A cost-benefit analysis contrasts financial benefits with financial costs
 - Costs are one time capital expense
 - Benefits accrue overtime
- Return on Investment (ROI)
 - $ROI = \text{Benefits}/\text{Cost}$
- Payback period
 - the length of time before accumulative benefits recover the costs of implementation

CSE870: Advanced Software Engineering (System Design) Cheng



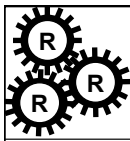
5.7 Architecture Evaluation and Refinement Computing Benefits (continued)

- Value may increase as quality attributes improve



- The net value of an improvement is the area under the curve

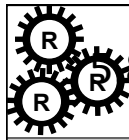
CSE870: Advanced Software Engineering (System Design) Cheng



5.7 Architecture Evaluation and Refinement Prototyping

- Some design decisions are best answered by prototyping
- **Prototype**: an executable model of the system built to answer specific questions about the system
- **Throw-away prototype**: meant to be discarded
- **Rapid prototyping**: progressively refine the prototype until it becomes the final system
- Potential risk: the customer may believe the operational prototype is the actual system and close to being finished

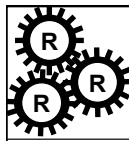
CSE870: Advanced Software Engineering (System Design) Cheng



8 Documenting Software Architectures

- System's architecture is vital to overall development and serves as the basis on decisions for:
 - Design
 - Quality assurance
 - Project management
- The SAD serves as the repository for design information and includes:
 - System overview
 - Views
 - Software units
 - Analysis data and results
 - Design rationale
 - Definitions, glossary, acronyms

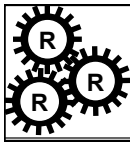
CSE870: Advanced Software Engineering (System Design) Cheng



5.8 Documenting Software Architectures Mappings among Views

- Structure of the system and intended measured attributes determine number and type of views to include in SAD
 - should at least include decomposition and execution view
- Design is collection of views; must show how views relate to one another

CSE870: Advanced Software Engineering (System Design) Cheng

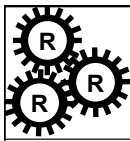


5.8 Documenting Software Architectures

Documenting Rationale

- **Document rationale:** outlining critical issues and trade-offs
- When to document the rationale behind decision:
 - Significant time spent on decision
 - Decision is critical
 - Decision is counterintuitive
 - Costly to change decision

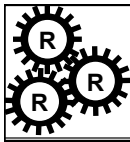
CSE870: Advanced Software Engineering (System Design) Cheng



5.9 Architecture Design Review

- Design review is an essential part of engineering practice
- SAD quality is evaluated in two ways:
 - **Validation:** making sure the design satisfies all of the customer's requirements (i.e., is this the right system?)
 - **Verification:** ensuring the design adheres to good design principles (i.e., are we building the system right?)

CSE870: Advanced Software Engineering (System Design) Cheng

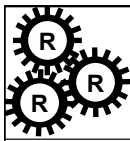


5.9 Architecture Design Review

Validation

- Several key people included in review:
 - The analyst(s) who helped define the system requirements
 - The system architect(s)
 - The program designer(s) for this project
 - A system tester
 - A system maintainer
 - A moderator
 - A recorder
 - Other interested developers not otherwise involved in this project

CSE870: Advanced Software Engineering (System Design) Cheng

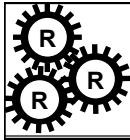


5.9 Architecture Design Review

Verification

- Judge whether it adheres to good design principles:
 - Is the architecture modular, well structured, and easy to understand?
 - Can we improve the structure and understandability of the architecture?
 - Is the architecture portable to other platforms?
 - Are aspects of the architecture reusable?
 - Does the architecture support ease of testing?
 - Does the architecture maximize performance, where appropriate?
 - Does the architecture incorporate appropriate techniques for handling faults and preventing failures?
 - Can the architecture accommodate all of the expected design changes and extensions that have been documented?

CSE870: Advanced Software Engineering (System Design) Cheng

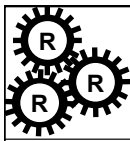


5.9 Architecture Design Review

Verification (continued)

- **Active design review:** exercise the design document by using it in ways the developers will use the final document in practice
- **Passive review process:** reading the documentation and looking for problems

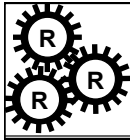
CSE870: Advanced Software Engineering (System Design) Cheng



5.10 Software Product Lines

- Organizations can find success by reusing their expertise and software assets across families of related products
- The corporate strategy for designing and developing the related products is based on the reuse of elements of a common **product line**
- A distinguishing feature of building a product line is the treatment of the derived products as a **product family**; their simultaneous development is planned from the beginning
- The family's commonalities are described as a collection of reusable assets (including requirements, designs, code, and test cases), all stored in a **core asset base**

CSE870: Advanced Software Engineering (System Design) Cheng

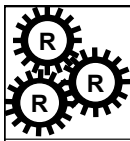


5.10 Software Product Lines

Core Asset Base

- Candidate elements in a core asset base:
 - Requirements
 - Software architecture
 - Models and analysis results
 - Software units
 - Testing
 - Project planning
 - Team organization

CSE870: Advanced Software Engineering (System Design) Cheng

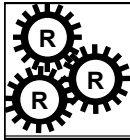


5.10 Software Product Lines

Strategic Scoping

- Product lines are based not just on commonalities among products but also on the best way to exploit them
 - First, employ strategic business planning to identify the family of products we want to build, using knowledge and good judgment to forecast market trends and predict the demand for various products
 - Second, **scope** the plans, so that the focus is on products that have enough in common to warrant a product-line approach to development. That is, the cost of developing the (common) product line must be more than offset by the savings we expect to accrue from deriving family members from the product line

CSE870: Advanced Software Engineering (System Design) Cheng

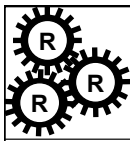


5.10 Software Product Lines

Sidebar 5.8 Product-line Productivity

- CelsiusTech AB, a Swedish naval defense contractor, motivated by desperation, transitioned from custom to product-line development. In 1985, the company, then Philips Elektronikindustier AB, was awarded two major contracts simultaneously, one for the Swedish Navy and one for the Danish Navy.
 - senior managers questioned whether they would be able to meet the demands of both contracts, particularly the promised (and fixed) schedules and budgets, using the company's current practices and technologies.
- Development of the product line and the first system were initiated at the same time; development of the second system started six months later. The two systems plus the product line were completed using roughly the same amount of time and staff that was needed previously for a single product. Subsequent products had shorter development timelines. On average, 70–80 percent of the seven systems' software units were product-line units (re)used as is.

CSE870: Advanced Software Engineering (System Design) Cheng

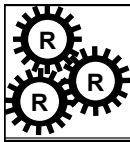


5.10 Software Product Lines

Advantages of Product-Line Architecture

- A product lines promotes planned modifiability
- Examples of product-line variability:
 - Component replacements
 - Component specializations
 - Product-line parameters
 - Architecture extensions and retractions

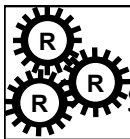
CSE870: Advanced Software Engineering (System Design) Cheng



Organizations and Products in SW Product Line Hall of Fame

- ...
- General Motors Powertrain (GMPT)
- **Hewlett Packard :**
 - printer firmware:
 - Resource savings:
 - 1/4 of the staff
 - in 1/3 of the time, and
 - with 1/25 the number of bugs of earlier products.
- Lucent
- Market Maker (stock market tracker/analyzer)
- **Nokia:**
 - Currently 32 different phones are manufactured
 - covering six different protocol standards, a wide variety of functional features and capabilities, different user interface designs, and many platforms and environments.
 - World's largest phone manufacturer

CSE870: Advanced Software Engineering (System Design) Cheng

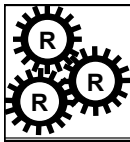


5.10 Software Product Lines

Sidebar 5.9 Generative Software Development

- **Generative software development** is a form of product-line development that enables products to be generated automatically from specifications
- The domain engineer defines a **domain-specific language (DSL)** that application engineers then use to specify products to be generated
- Lucent developed several product lines and generative tools for customizing different aspects of its 5ESS telephone switch

CSE870: Advanced Software Engineering (System Design) Cheng

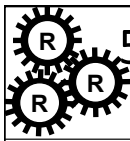


5.10 Software Product Lines

Product-Line Evolution

- Key contributor to product-line success is having a product-line mindset
 - Company's primary focus is development and evolution of product-line assets as opposed to individual products
 - Changes made to improve capability to derive products
 - Backwards capability

CSE870: Advanced Software Engineering (System Design) Cheng

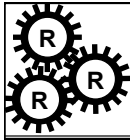


5.11 Information System Example

Piccadilly System

- What might be a suitable architecture for the Piccadilly systems?
- Key components
 - A repository of information
 - Address multiple heterogeneous queries
- A typical reference architecture for an information system
 - n-tiered client-server architecture

CSE870: Advanced Software Engineering (System Design) Cheng

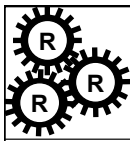


5.12 Real-Time Example

Ariane-5 Failure

- Inquiry found that the Ariane program had a “culture...of only addressing random hardware failures” and assuming the software was correct
- Hardware failures are independent of one another
- Software faults tend to be logical
 - All redundant components will have the same faults
- Redundancy in Ariane-5 is likely to recover

CSE870: Advanced Software Engineering (System Design) Cheng



5.13 What This Chapter Means For You

- Systems need to be designed based on carefully expressed requirements
- Design begins with a high-level architecture, where architectural decisions are based not only on system functionality and required constraints but also on desirable attributes and the long-term intended use of the system (including product lines, reuse, and likely modification)
- Keep in mind several characteristics of good architecture as you go, including appropriate user interfaces, performance, modularity, security, and fault tolerance
- The goal is not to design the ideal software architecture for a system, because such an architecture might not even exist. Rather, the goal is to design an architecture that meets all of the customer's requirements while staying within the cost and schedule constraints

CSE870: Advanced Software Engineering (System Design) Cheng