OMSE 532 - Week 3 Architectural Qualities

Stuart Faulk
University of Oregon

Overview

- Categorizing architectural qualities
- Quality scenarios

Expected Outcome

- Description of overall organizing principles
- Rationale
  - Connect design decisions to requirements/goals
  - Optional description of alternatives
- Components and relations
  - Including major interfaces
  - In (just) enough detail to serve as basis for feasibility and cost studies
- Focus on questions and Issues
  - What design issues emerge? Requirements questions?
  - What questions does the problem raise about developing an appropriate architecture?

Spam Filter Design Process

Methodology independent view implied by purpose and context

1. Creating the business case for the system
   – Which business goals are affected by the architecture?
2. Understanding the requirements
   – What are the quality requirements?
   – Relation to: stakeholder, business goal, priority?
3. Designing the architecture
   – Which components, relations, and interfaces? Decomposition criteria?
   – Which tradeoffs maximize requirements/goals?
4. Representing and communicating the architecture
   – Audience of and purpose for each view?
   – What should be represented and how?
5. Analyzing or evaluating the architecture
   – What evaluation criteria should be used?
6. Implementing the system based on the architecture
7. Ensuring the implementation conforms to the architecture

Evaluating Solutions

- Sufficient for purpose?
  – Basis for cost/feasibility estimate
    - Can we evaluate the schedule/cost consequences of architectural choices?
  – First step to product architecture
- Specific goals
  – Domain fit: standards-based email
  – Constraints: laptop / server / PDA / …
  – Change/add filtering approaches on the fly
  – Customization and evolution of product line

Placement Alternatives
A Major Issue

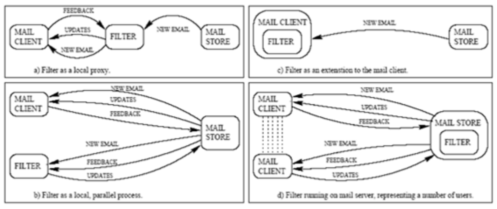
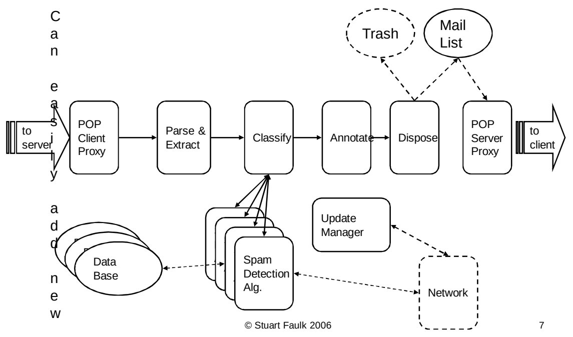

FEEDBACK
UPDATES
NEW EMAIL
MAIL CLIENT
FILTER
MAIL STORE
a) Filter as a local proxy.
c) Filter as an extension to the mail client.
b) Filter as a local, parallel process.
d) Filter running on mail server, representing a number of users.

Fig. 1. Possible placements of filter.

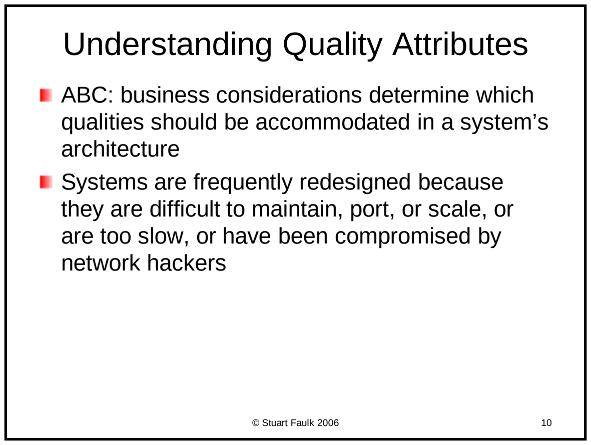
Example: Simple Functional Solution POP Proxy



A Note on Views

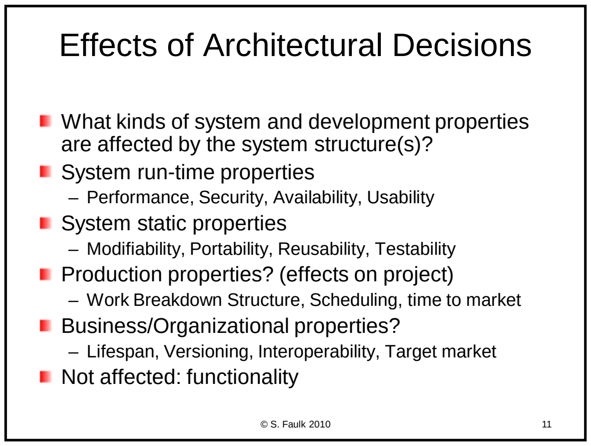
- Architectural descriptions are models
  - They emphasize some characteristics and suppress others
- No single model is good for everything
  - An architectural sketch for cost/feasibility is unlikely to be sufficient for performance studies
- But models must be consistent
  - It is easy to end up with “views” that cannot be reconciled

Architectural Qualities



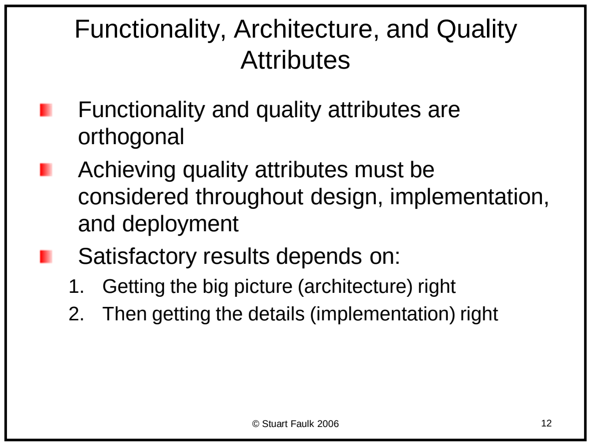
Understanding Quality Attributes

- ABC: business considerations determine which qualities should be accommodated in a system's architecture
- Systems are frequently redesigned because they are difficult to maintain, port, or scale, or are too slow, or have been compromised by network hackers



Effects of Architectural Decisions

- What kinds of system and development properties are affected by the system structure(s)?
- System run-time properties
  - Performance, Security, Availability, Usability
- System static properties
  - Modifiability, Portability, Reusability, Testability
- Production properties? (effects on project)
  - Work Breakdown Structure, Scheduling, time to market
- Business/Organizational properties?
  - Lifespan, Versioning, Interoperability, Target market
- Not affected: functionality



Functionality, Architecture, and Quality Attributes

- Functionality and quality attributes are orthogonal
- Achieving quality attributes must be considered throughout design, implementation, and deployment
- Satisfactory results depends on:
  1. Getting the big picture (architecture) right
  2. Then getting the details (implementation) right

Functionality, Architecture, and Quality Attributes

- Ex: Performance depends on
  - How much inter-component communication is necessary (Arch)
  - What functionality has been allocated to each component (Arch)
  - How shared resources are allocated (Arch)
  - The choice of algorithms to implement functionality (Non-arch)
  - How algorithms are coded (Non-arch)

Terminology

- Avoid “functional” and non-functional" classification
- Behavioral Requirements - Any information necessary to determine if the run-time behavior of a given implementation constitutes an acceptable system
  - All quantitative constraints on the system's run-time behavior
  - Other objective measures (safety, performance, etc)
  - *In theory* all can be validated by observing the running system and measuring the results
- Developmental Quality Attributes - Any constraints on the system's static construction
  - Testability, ease of change (mutability), maintainability, reusability
  - Measures of these qualities are necessarily relativistic (I.e., in comparison to something else)

Behavioral vs. Developmental

Behavioral (observable)	**Developmental Qualities**
Performance	Modifiability
Security	Portability
Availability (fault-tolerance)	Reusability
Security	Integrability
Usability	Testability
Properties resulting from the properties of components, connectors and interfaces that exist at run time.	Properties resulting from the properties components, connectors and interfaces that exist at design time *whether or not they have any distinct run-time manifestation*.

Behavioral vs. Developmental (2)

Behavioral (observable)	Developmental Qualities
Performance	Modifiability
Security	Portability
Availability (fault-tolerance)	Reusability
Usability (responsiveness?)	Integrability
	Testability

- Usefully viewed as distinct concerns
  - Visible at different times
  - Can focus on one at a time
- Often not easy to separate in practice
  - Design time mechanisms often carried into run-time structures
  - Real separation requires careful engineering
  - Many mechanisms bind more than one attribute at a time or abstract from what we want to control. Examples?
- The "art" of design includes finding structures that:
  - Address multiple concerns concurrently and/or
  - Cleanly separate design from run-time constraints

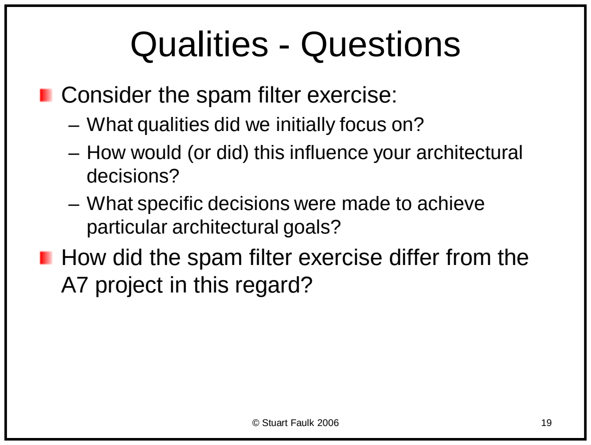
Additional Properties

- Business process qualities
  - Properties of the system that
    - Are affected by architectural decisions
    - Reflect the business context (business goals, environment, constraints)
  - Time to market, cost/benefit, projected life, target market, rollout schedule, legacy integration
  - Observable by viewing the business process around the development
- Design qualities
  - Conceptual integrity, correctness, completeness, feasibility

Architectural Design Process

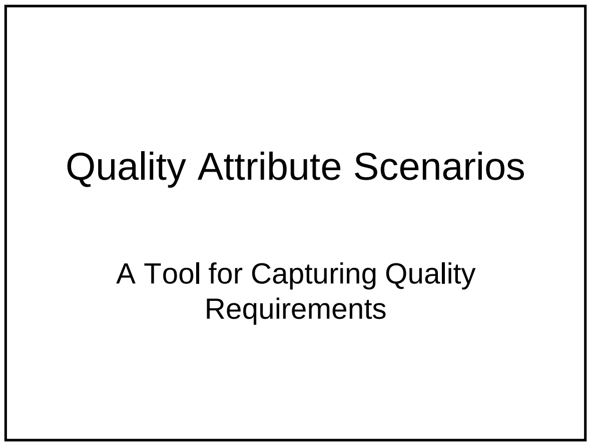
Methodology independent view implied by *purpose* and *context*

1. Creating the business case for the system
2. Understanding the (quality) requirements
   1. Identify & prioritize behavioral and quality attributes
   2. Characterize and communicate to stakeholders
3. Creating or selecting the architecture
4. Representing and communicating the architecture
5. Analyzing or evaluating the architecture
6. Implementing the system based on the architecture
7. Ensuring the implementation conforms to the architecture



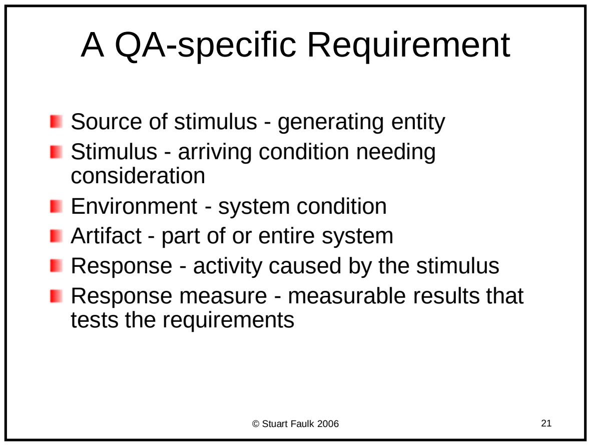
Qualities - Questions

- Consider the spam filter exercise:
  - What qualities did we initially focus on?
  - How would (or did) this influence your architectural decisions?
  - What specific decisions were made to achieve particular architectural goals?
- How did the spam filter exercise differ from the A7 project in this regard?



Quality Attribute Scenarios

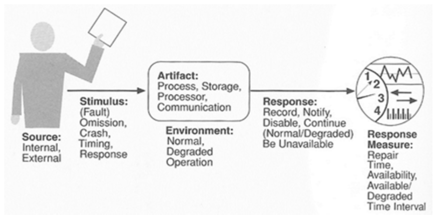
A Tool for Capturing Quality Requirements



A QA-specific Requirement

- Source of stimulus - generating entity
- Stimulus - arriving condition needing consideration
- Environment - system condition
- Artifact - part of or entire system
- Response - activity caused by the stimulus
- Response measure - measurable results that tests the requirements

Quality Scenarios



Source: Internal, External
Stimulus: (Fault) Omission, Crash, Timing, Response
Artifact: Process, Storage, Processor, Communication
Environment: Normal, Degraded Operation
Response: Record, Notify, Disable, Continue (Normal/Degraded) Be Unavailable
Response Measure: Repair Time, Availability, Available/ Degraded Time Interval
1
2
3
4

- Options (Chinese menu) for each type of quality attribute
- Used to “generate” general scenarios (must be edited)
- Focus on making sure all possibilities are considered

Availability Table*

- Source: internal, external
- Stimulus: type of fault
- Artifact: processors, channels, storage, process
- Environment: normal, degraded
- Response: logging, notification, switching to backup, restart, shutdown
- Measure: availability, repair time, required uptime

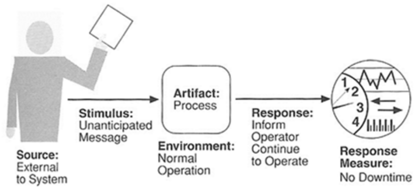
**Have general scenario generation table for each quality*

Availability Table*

- Source: internal, external
- Stimulus: type of fault
- Artifact: processors, channels, storage, process
- Environment: normal, degraded
- Response: logging, notification, switching to backup, restart, shutdown, continue
- Measure: availability, repair time, required uptime

**General scenario from choosing parts*

Quality Scenarios - Concrete


Source: External to System
Stimulus: Unanticipated Message
Artifact: Process
Environment: Normal Operation
Response: Inform Operator Continue to Operate
Response Measure: No Downtime

- Must supply real values for each part
- Should be system specific
- Realistically, requires some rewriting of elements from "general" scenario to make sense

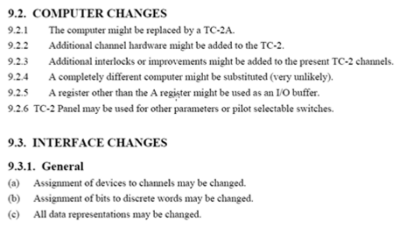
Properties of Good QASs

- Values of each part are reasonable and useful
- The values must be consistent with one another
  - Values work together to describe a specific kind of quality requirement
  - E.g., for a performance requirement, all attributes address aspects related to same performance measure
- The set of choices actually specify a quality requirement
- The resulting scenario must specify a relevant system requirement
- To achieve this, need to be clear on what you are trying to specify

Applying Scenarios

Examples from A-7:
Modifiability
Extensibility/Portability
Performance

Modifiability Example (A-7E)



9.2. COMPUTER CHANGES

9.2.1 The computer might be replaced by a TC-2A.
9.2.2 Additional channel hardware might be added to the TC-2.
9.2.3 Additional interlocks or improvements might be added to the present TC-2 channels.
9.2.4 A completely different computer might be substituted (very unlikely).
9.2.5 A register other than the A register might be used as an I/O buffer.
9.2.6 TC-2 Panel may be used for other parameters or pilot selectable switches.

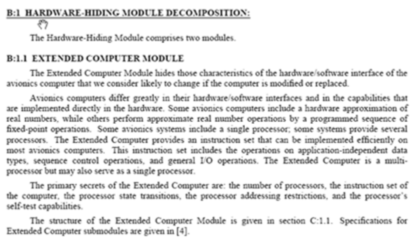
9.3. INTERFACE CHANGES

9.3.1. General

(a) Assignment of devices to channels may be changed.
(b) Assignment of bits to discrete words may be changed.
(c) All data representations may be changed.

A-7E Requirements Spec lists likely change (View as QAS where only the "Stimulus" changes)

Map Changes to Modules



B:1 HARDWARE-HIDING MODULE DECOMPOSITION:

The Hardware-Hiding Module comprises two modules.

B:1.1 EXTENDED COMPUTER MODULE

The Extended Computer Module hides those characteristics of the hardware/software interface of the avionics computer that we consider likely to change if the computer is modified or replaced.

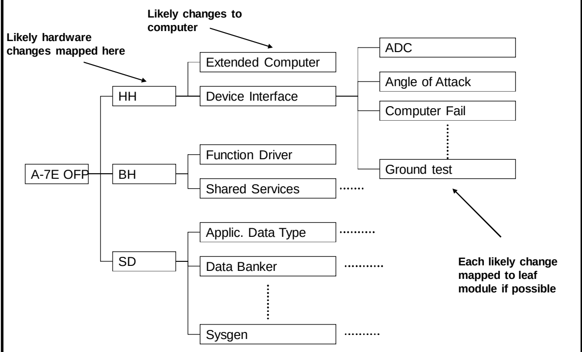
Avionics computers differ greatly in their hardware/software interfaces and in the capabilities that are implemented directly in the hardware. Some avionics computers include a hardware approximation of real numbers, while others perform approximate real number operations by a programmed sequence of fixed-point operations. Some avionics systems include a single processor; some systems provide several processors. The Extended Computer provides an instruction set that can be implemented efficiently on most avionics computers. This instruction set includes the operations on application-independent data types, sequence control operations, and general I/O operations. The Extended Computer is a multi-processor but may also serve as a single processor.

The primary secrets of the Extended Computer are: the number of processors, the instruction set of the computer, the processor state transitions, the processor addressing restrictions, and the processor's self-test capabilities.

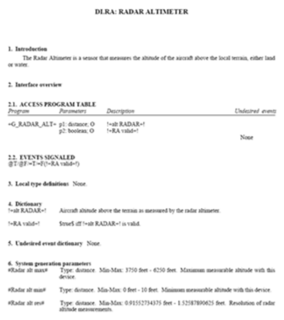
The structure of the Extended Computer Module is given in section C:1.1. Specifications for Extended Computer submodules are given in [4].

Design of module structure (module decomposition) allocates changes to modules

A-7E Module Structure



Define Interface for Leaf Modules



DLRA: RADAR ALTIMETER

1. Introduction

The Radar Altimeter is a sensor that measures the altitude of the aircraft above the local terrain, either land or water.

2. Interface overview

2.1. ACCESS PROGRAM TABLE

Program	*Parameters*	*Description*	*Undesired events*
+G_RADAR_ALT+	p1: distance; O	!+alt RADAR+!	
	p2: boolean; O	!+RA valid+!	
			None

2.2. EVENTS SIGNALED
@T/@F/=T/=F(!+RA valid+!)

3. Local type definitions None.

4. Dictionary

!+alt RADAR+!	Aircraft altitude above the terrain as measured by the radar altimeter.
!+RA valid+!	$true$ iff !+alt RADAR+! is valid.

5. Undesired event dictionary None.

6. System generation parameters

#Radar alt max#	Type: distance. Min-Max: 3750 feet - 6250 feet. Maximum measurable altitude with this device.
#Radar alt min#	Type: distance. Min-Max: 0 feet - 10 feet. Minimum measurable altitude with this device.
#Radar alt res#	Type: distance. Min-Max: 0.91552734375 feet - 1.52587890625 feet. Resolution of radar altitude measurements.

- Hides likely changes
- Defines everything necessary to implement the module
  - Services
  - Parameters
  - Timing constraints
  - Synchronization events
- Basis for
  - Abstract devices and abstract types
  - Work assignments
  - Understandability (study independently)
  - Unit testing

© Stuart Faulk 2006 31

Module Structure Evaluation Using Scenarios

- Procedure
  - "Play" each scenario against the module architecture
  - Review for understandability, etc.
- Completeness (internal)
  - Do the submodules of each module partition its secrets?
  - Is every change scenario the responsibility of some module?
- Modifiability
  - Is every likely change hidden by some module?
  - Are only aspects of the system that are very unlikely to change embedded in the module structure?
  - For each leaf module, are the module' s secrets revealed by it' s access programs?

© Stuart Faulk 2006 32

Performance

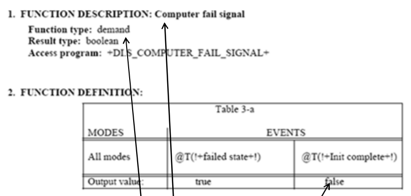
© Stuart Faulk 2006 33

Performance Table

- Source: external, internal
- Stimulus: event arrival pattern
  - periodic
  - stochastic
  - sporadic
- Artifact: system services
- Environment: normal, overload
- Response: change in mode?
- Measure: latency, deadline, throughput, jitter, miss rate, data loss

34

Required Demand Function



1. FUNCTION DESCRIPTION: Computer fail signal

Function type: demand
Result type: boolean
Access program: +DIS_COMPUTER_FAIL_SIGNAL+

2. FUNCTION DEFINITION:

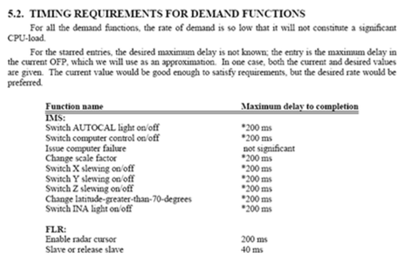
Table 3-a

MODES	EVENTS	
All modes	@T(!+failed state+!)	@T(!+Init complete+!)
Output value:	true	false

- One for each system output
- Defines the arrival pattern and environment (system state)
- Defines the required response (output value)

© Stuart Faulk 2006 35

Timing Requirements



5.2. TIMING REQUIREMENTS FOR DEMAND FUNCTIONS

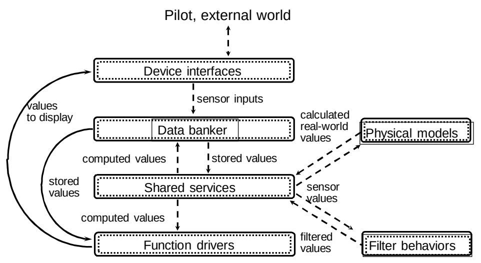
For all the demand functions, the rate of demand is so low that it will not constitute a significant CPU-load.

For the starred entries, the desired maximum delay is not known; the entry is the maximum delay in the current OFP, which we will use as an approximation. In one case, both the current and desired values are given. The current value would be good enough to satisfy requirements, but the desired rate would be preferred.

Function name	Maximum delay to completion
IMS:	
Switch AUTOCAL light on/off	*200 ms
Switch computer control on/off	*200 ms
Issue computer failure	not significant
Change scale factor	*200 ms
Switch X slewing on/off	*200 ms
Switch Y slewing on/off	*200 ms
Switch Z slewing on/off	*200 ms
Change latitude-greater-than-70-degrees	*200 ms
Switch INA light on/off	*200 ms
FLR:	
Enable radar cursor	200 ms
Slave or release slave	40 ms

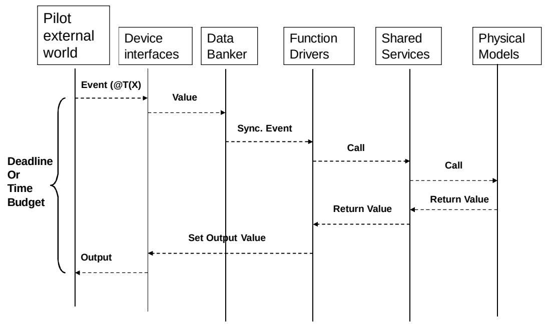
© Stuart Faulk 2006 36

Data Flow View



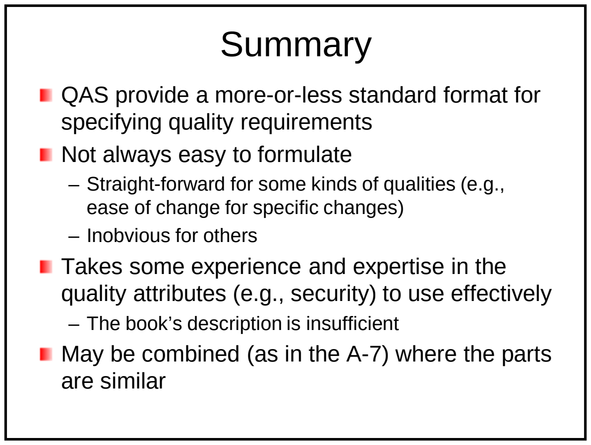
Based on CMU/SEI tutorial for Software Architecture in Practice

Consistent Design Pattern



Evaluation

- Each process must meet its deadline
- Together, must be able to produce feasible schedule
  - Processes share common resources
  - Additional scenarios must account for requirements on the whole set



Summary

- QAS provide a more-or-less standard format for specifying quality requirements
- Not always easy to formulate
  - Straight-forward for some kinds of qualities (e.g., ease of change for specific changes)
  - Inobvious for others
- Takes some experience and expertise in the quality attributes (e.g., security) to use effectively
  - The book's description is insufficient
- May be combined (as in the A-7) where the parts are similar

Questions?