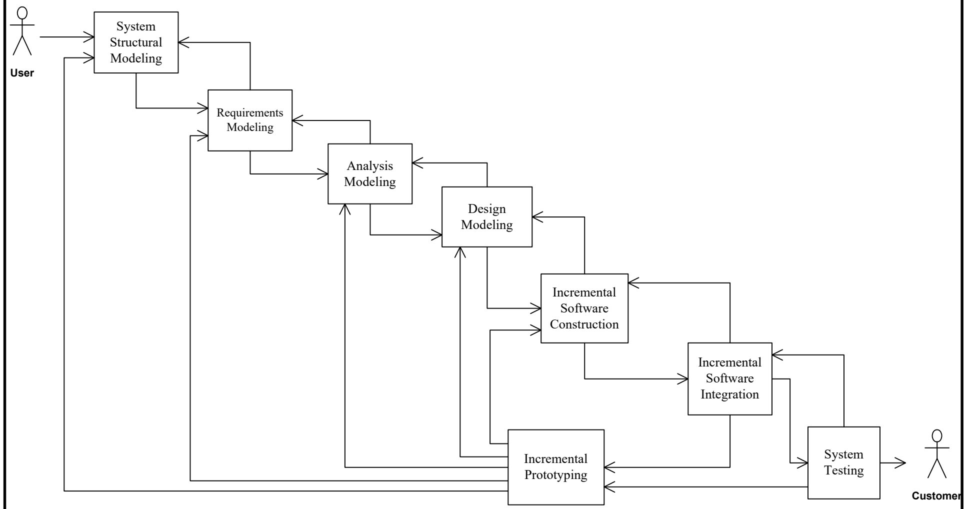
SWE 760

Lecture 13 – System and Software Quality Attributes for Real-Time Embedded Systems

Reference:

H. Gomaa, Chapters 16 - *Real-Time Software Design for Embedded Systems*, Cambridge University Press, 2016

Figure 4.1 COMET/RTE life cycle model



User

System Structural Modeling

Requirements Modeling

Analysis Modeling

Design Modeling

Incremental Software Construction

Incremental Software Integration

Incremental Prototyping

System Testing

Customer

System and Software Quality Attributes

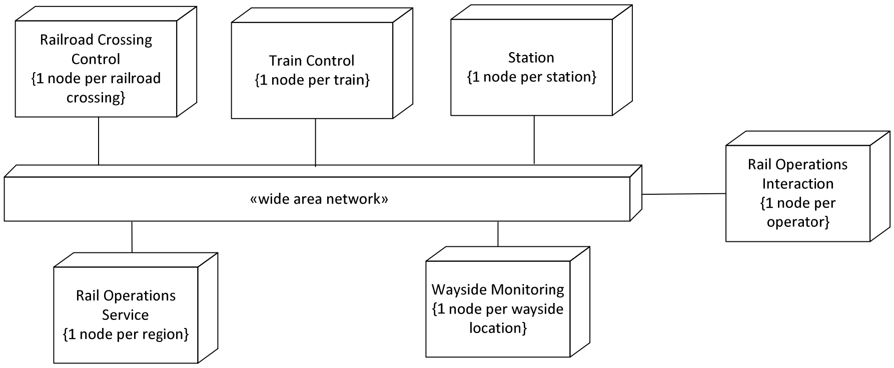
- Address non-functional requirements
- System (hardware + software) Quality Attributes
  - Scalability
  - Performance
  - Availability
  - Safety
  - Security
- Software Quality Attributes
  - Maintainability
  - Modifiability
  - Testability
  - Traceability
  - Reusability

Scalability

- Extent to which the system is capable of growing after its initial deployment
- System needs to be designed in such a way that it is capable of growth
- Distributed component-based software architecture
  - Much more capable of scaling upwards than a centralized design
  - Components are designed such that multiple instances of each component can be deployed to different nodes in a distributed configuration

Example of Scalability

Figure 16.1 Scaleup in Light Rail System

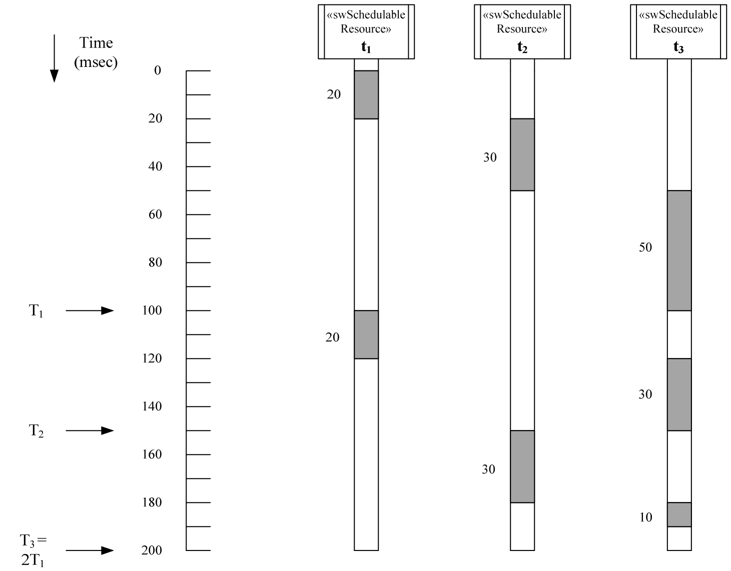

Railroad Crossing Control {1 node per railroad crossing}
Train Control {1 node per train}
Station {1 node per station}
Rail Operations Interaction {1 node per operator}
«wide area network»
Rail Operations Service {1 node per region}
Wayside Monitoring {1 node per wayside location}

Performance

- **Performance analysis**
  - Quantitative analysis of a *real-time* software design
  - Conceptually executing on a given hardware configuration
  - With a given external workload applied to it
- **Performance modeling**
  - Abstraction of the real computer system behavior
  - Developed for the purpose of gaining greater insight into the performance of the system
  - Whether or not the system actually exist
  - E.g., simulation modeling, real-time scheduling

Example of Performance

Figure 17.1 Timing diagram for tasks executing on a single processor system


«swSchedulable Resource» t_1
«swSchedulable Resource» t_2
«swSchedulable Resource» t_3
Time (msec)
0
20
40
60
80
100
120
140
160
180
200
T_1
T_2
$T_3 = 2T_1$
20
20
30
30
50
30
10

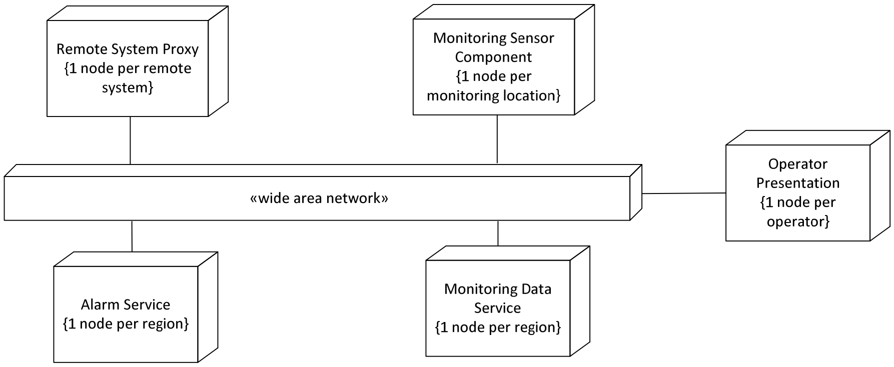
Availability

- Extent to which system is available for operational usage
  - Addresses system failure
  - E.g., system must be operational for 99% of time
- Fault tolerant systems
  - E.g., Triple redundancy and voting systems
- Hot standby, e.g., backup server in Banking system
- Software design
  - Systems without single points of failure
  - Distributed component-based software architectures
    - Deployed to multiple nodes
  - If a single node goes down
    - System can operate in a degraded mode.

Example of Availability

Figure 16.2 Example of system without single point of failure

- Minimize system failure
  - No single point of failure


Remote System Proxy {1 node per remote system}
Monitoring Sensor Component {1 node per monitoring location}
«wide area network»
Operator Presentation {1 node per operator}
Alarm Service {1 node per region}
Monitoring Data Service {1 node per region}

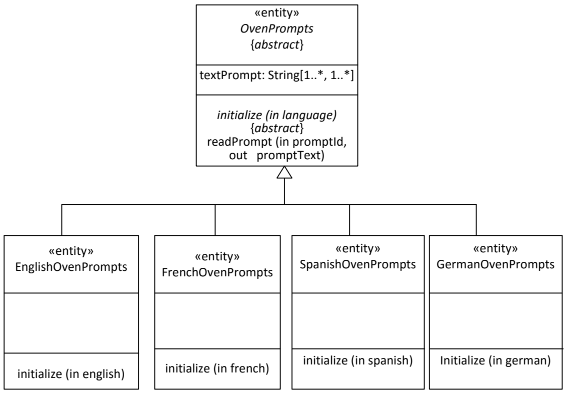
Safety

- Goal of System Safety: accident prevention
- Proactively identifying, assessing, and eliminating or controlling safety-related hazards, to acceptable levels, can achieve accident prevention (FAA)
- Hazard
  - A condition, event, or circumstance that could lead to or contribute to an unplanned or undesired event (FAA)
- Safety critical system
  - Safety-related hazards identified during requirements specification
  - Software design must detect hazards and take appropriate action
- Examples of safety requirements
  - Railroad Crossing Control System (Chapter 20),
    - Barrier must be lowered within a pre-specified time
  - Light Rail Control System
    - Train must slow down to a stop if an obstacle is detected

Modifiability

- Extent to which software is capable of being modified during and after initial development
- Design for Change,
  - e.g., Oven Prompt class with language specific subclasses

Figure 16.3 Example of modifiability - abstract Oven Prompts class and language specific subclasses



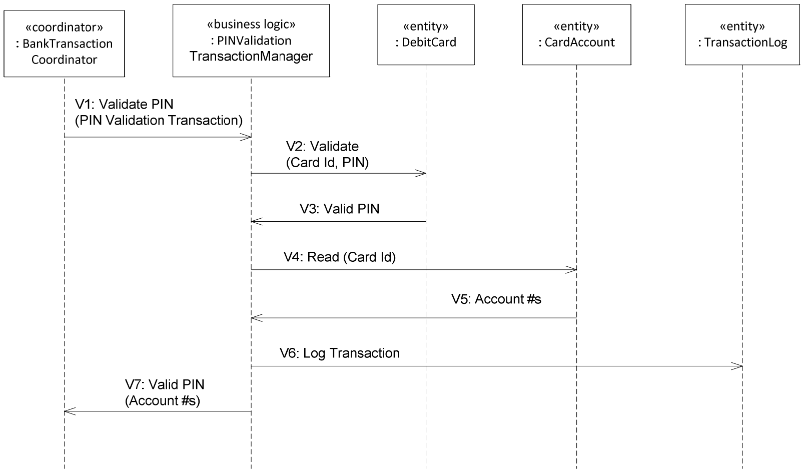
Testability

- Extent to which software is capable of being tested during and after its initial development
- During Requirements Phase
  - Develop functional (black box) test cases
  - Develop test cases from use case descriptions
- During Software Architectural Design
  - Develop integration test cases
  - Test interfaces between communicating components
- Scenario based testing
  - Develop integration test cases from interaction scenarios sequence or communication diagrams

Determine scenario to test from Sequence Diagram

Determine testing scenario from sequence diagram

Figure 21.15 Sequence diagram: Banking Service – Validate PIN use case


«coordinator»
: BankTransaction Coordinator
«business logic»
: PINValidation TransactionManager
«entity»
: DebitCard
«entity»
: CardAccount
«entity»
: TransactionLog
V1: Validate PIN (PIN Validation Transaction)
V2: Validate (Card Id, PIN)
V3: Valid PIN
V4: Read (Card Id)
V5: Account #s
V6: Log Transaction
V7: Valid PIN (Account #s)

Traceability

- Extent to which artifacts of each phase can be traced back to products of previous phases
- Build traceability into software development method
- Software requirements – use case model
- Use case based interaction diagrams
  - Determines objects required to realize each use case
  - Determine sequence of interactions between objects
- Software architecture
  - Integrate use case based interaction diagrams
- Impact Analysis
  - Determine impact of software change using traceability

Figure 16.4 Traceability analysis before and after change to introduce Oven Prompts object

Example of Traceability

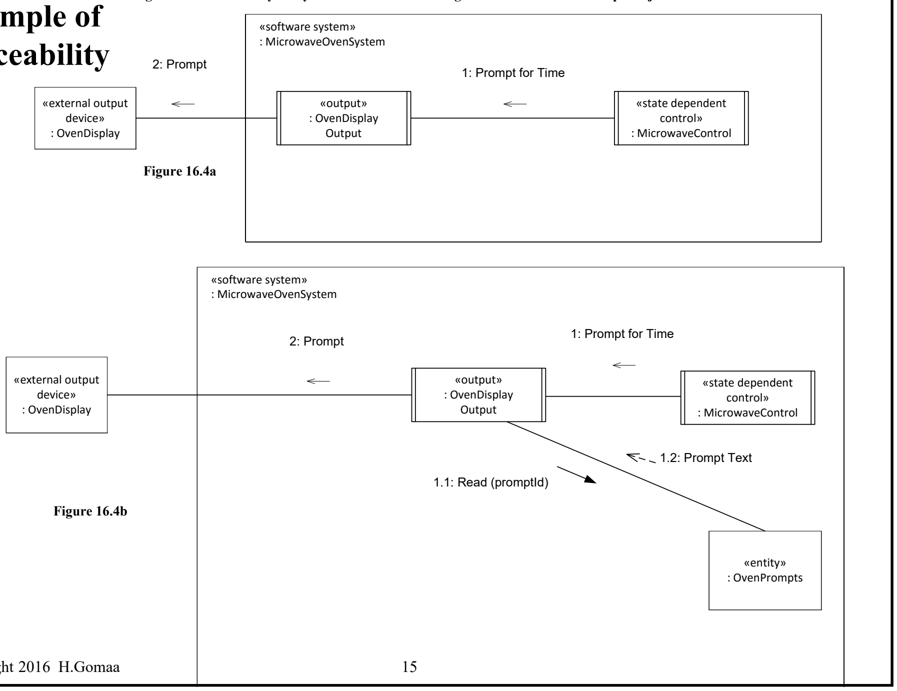
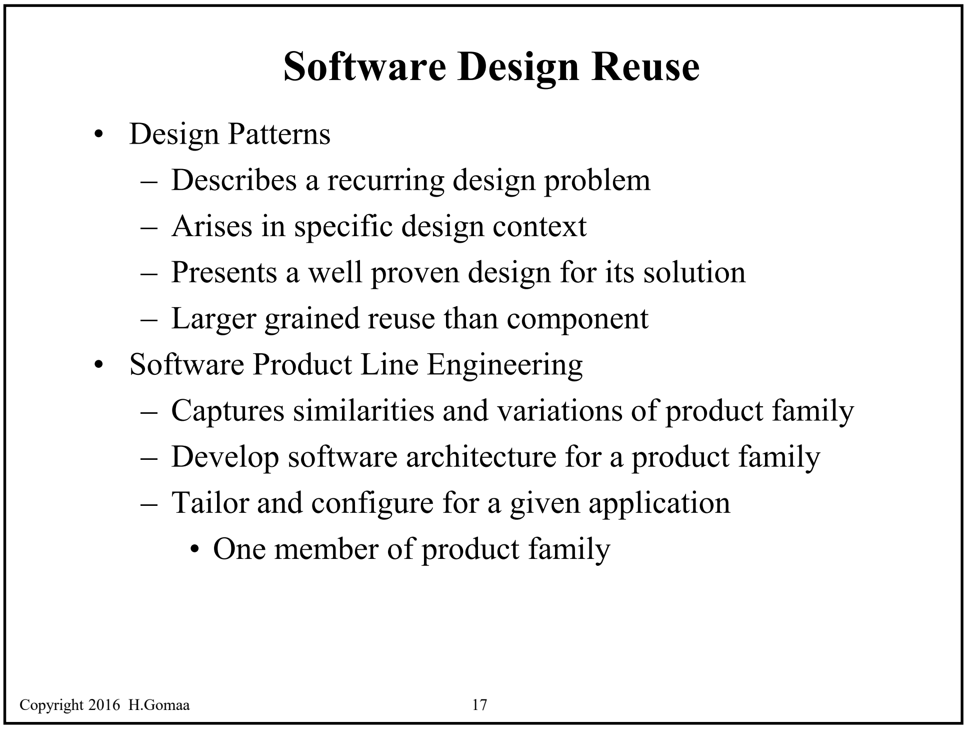


Figure 16.4a

Figure 16.4b

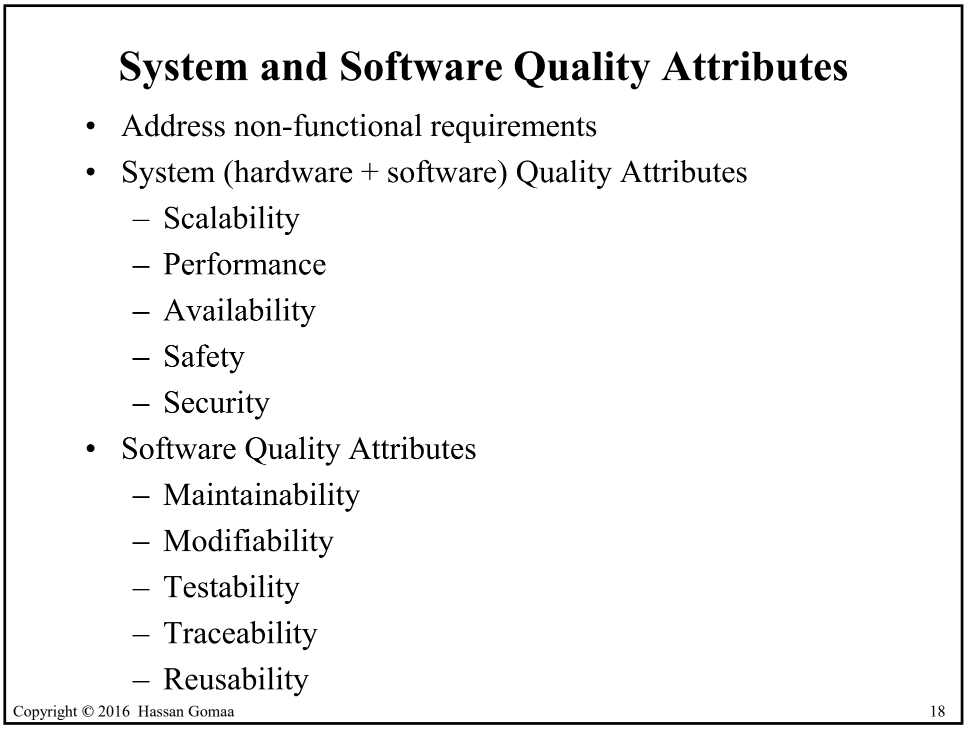
Reusability

- Extent to which software is capable of being reused
- Software Component Reuse
  - Library of reusable code components
    - May be functional or object-oriented
- Software Design reuse
  - Reuse components and their interconnections
- Architecture reuse
  - Large grained reuse
  - Focuses on requirements and design
  - Much greater potential than component reuse
- Generic architecture
  - One architecture for the application domain
  - Manually adapted (tailored) for a specific application



Software Design Reuse

- Design Patterns
  - Describes a recurring design problem
  - Arises in specific design context
  - Presents a well proven design for its solution
  - Larger grained reuse than component
- Software Product Line Engineering
  - Captures similarities and variations of product family
  - Develop software architecture for a product family
  - Tailor and configure for a given application
    - One member of product family



System and Software Quality Attributes

- Address non-functional requirements
- System (hardware + software) Quality Attributes
  - Scalability
  - Performance
  - Availability
  - Safety
  - Security
- Software Quality Attributes
  - Maintainability
  - Modifiability
  - Testability
  - Traceability
  - Reusability