SWINBURNE UNIVERSITY OF TECHNOLOGY

SWE30003
Software Architectures and Design

Lecture 11
Documenting Software Designs, and
Evaluating Software Design

1

Logistical matters

- Weekly submissions – A & Q
  - Week 2: 362 and 341 out of 459;
  - Week 3: 397 and 375 out of 459;
  - Week 4: 399 and 390 out of 459;
  - Week 5: 380 and 372 out of 453;
  - Week 6: 389 and 382 out of 453;
  - Week 7: 362 and 356 out of 452;
  - Week 8: 362 and 349 out of 452;
  - Week 9: 371 and 364 out of 452;
  - Week 10: 346 and 341 out of 450;
  - Week 11: xxx and yyy out of 450; **THE END**.
- Assignment 3: Questions …

2

2

Question to Answer – Week 10

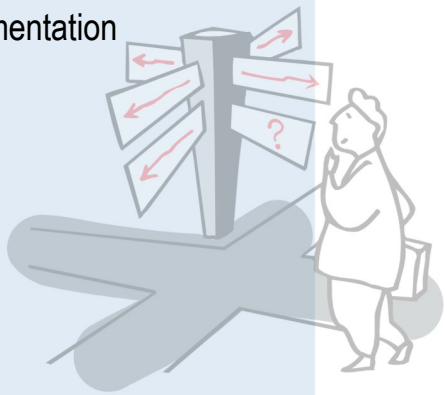
Discuss the relationships (commonalities and differences) between the concepts of software service, SOAP web service, REST web service, and microservice.

3

3

Outline

- Design Documentation
  - Purpose of Documentation
  - Views
  - Notations
- Design Evaluation



4

4

Principal References

- Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice (4th Edition)*, Addison-Wesley, 2021, Chapters 21 and 22. *OR*

  Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice (3rd Edition)*, Addison-Wesley, 2013, Chapters 18 and 21.

- Mary Shaw and David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996, Chapters 3 and 6.

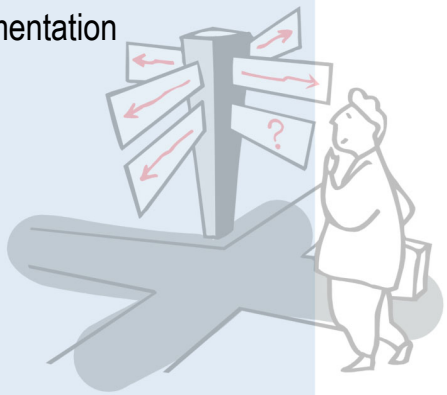
- *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (IEEE Std. 1471-2000), September 2000.

5

5

Outline

- Design Documentation
  - Purpose of Documentation
  - Views
  - Notations
- Design Evaluation



6

6

Principal Focus

- Why do we want to document software designs?
  - ☞ *special focus on architectural design*
- Who will be interested in reading documented designs?

7

7

The role of Software Design

- Bridging the gap between requirements and implementation
- Assisting software engineers to develop software solutions to a given set of requirements
  - ☐ to enable *early verification* of software solutions
  - ☐ to produce a *blue-print* for implementation
- Enhancing the *communication* between a project's stakeholders

8

8

Purposes of Architecture

- Expression of the system and its evolution
- Communication among the system stakeholders
- Evaluation and comparison of architectures in a consistent manner
- Planning, managing, and executing the activities of system development
- Expression of the persistent characteristics and supporting principles of a system to guide acceptable change
- Verification of a system implementation's compliance with an architectural description
- Recording contributions to the body of knowledge of software-intensive systems architecture

— Source: IEEE Standard 1471

9

9

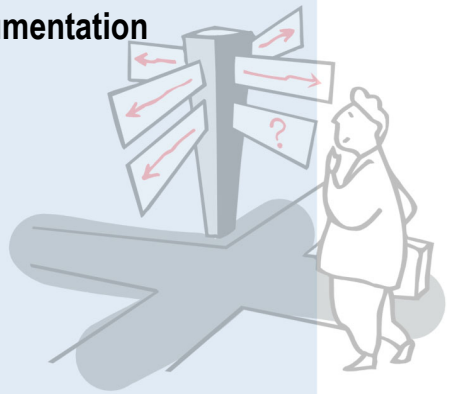
☞ *Any form of architectural documentation must address at least one of the purposes/goals of software architectures!*

10

10

Outline

- Design Documentation
  - **Purpose of Documentation**
  - Views
  - Notations
- Design Evaluation



11

11

Purpose of Documentation

Communication: communicating ideas about possible software solutions to stakeholders for feedback etc.

- ☐ Who is the audience? What do they want/need to know?
- ☐ What means of communication do they understand?

☞ *Focus on the big-picture ideas!*

Specification: blue-print for implementation

- ☐ Purpose: a detailed design specification for implementers.
- ☐ Needs to avoid any form of ambiguity.

☞ *Focus on specific details!*

12

12

Stakeholders of Software Projects

- Customer
- End-User
- Developer
  - ☐ Maintainer
- Manager
- Business Analyst
- System Administrator

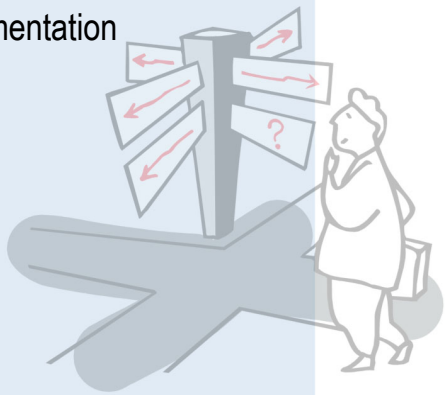
☞ *All have different interests and needs for documentation!*

13

13

Outline

- Design Documentation
  - Purpose of Documentation
  - **Views**
  - Notations
- Design Evaluation



14

14

Views

"A view is a representation of a coherent set of architectural elements, as written by and read by stakeholders. It consists of a representation of a set of [architectural] elements and the relationships amongst them."

— Len Bass et al., Software Architecture in Practice (2nd Edition), 2003.

☞ Views focus on specific aspects of a software system by using appropriate *abstractions*!

15

15

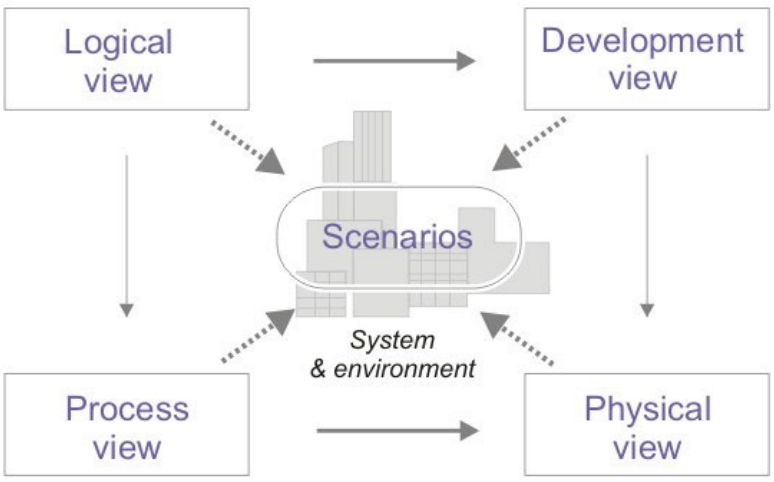
Categories of Views

- Component and Connector:
  - Focus on (run-time) *computational entities* (i.e. components, processing elements) and their means of *communication* (i.e. connectors).
  - *What are the main executing components and how do they interact?*
- Allocation:
  - Focus on computational entities and their *(physical) environment*.
  - *What processing node does each component execute on?*
- Module:
  - Focus on code units and their responsibilities
  - *What packaging mechanisms are used to specify/implement each architectural element? How are these "packages" inter-related?*

16

16

Kruchten's 4+1 View



17

17

Kruchten's 4+1 View (cont.)

- *Logical View* (Module View):
  - concerned with the *functionality* that a system provides to end-users
- *Process View* (Component & Connector View):
  - focus on system processes (i.e. components) and how they communicate; addresses concurrency, distribution, performance, scalability etc.
- *Development View* (Allocation View):
  - illustrates a system from a developers perspective; concerned with software management
- *Physical View* (Allocation View):
  - concerned with mapping of components onto processing and communication nodes

18

18

Kruchten's 4+1 View (cont.)

- Scenarios:
  - illustration of architectural solution using selected use cases,
  - describe sequences of interactions between system processes (i.e. processing elements),
  - used to **illustrate and verify** the architectural design,
  - serve as a starting point for tests and architectural prototypes.

☞ *In many situations, Kruchten's 4+1 view model is a good starting point for documenting architectural designs.*

19

19

However…

"A neat-looking drawing of some boxes, circles, and lines, laid out nicely in Powerpoint or Word, does not constitute an architecture."

— D'Souza & Wills

☞ There is a need to assist developers in avoiding ambiguous notations!

20

20

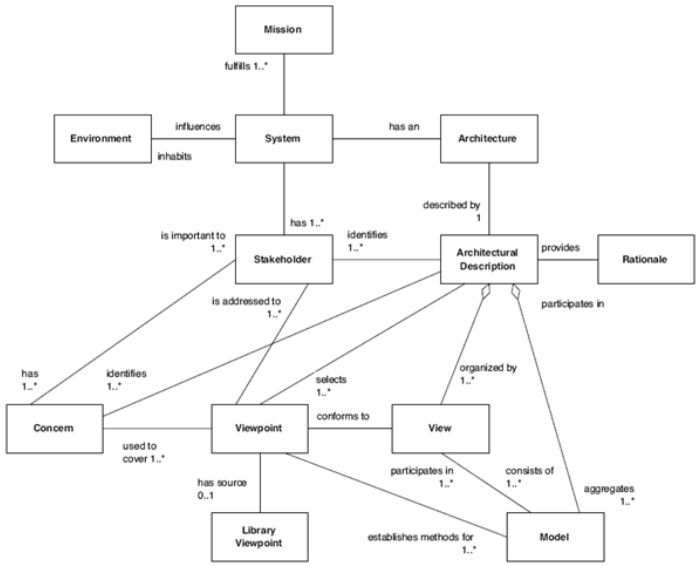
IEEE Standard 1471

- IEEE 1471 provides definitions and a *meta-model* for the description of software architectures.
- Assumes that an architecture exists to address to specific *stakeholder concerns*.
- Asserts that architectural descriptions are inherently multi-view
  - ☞ no single view captures all stakeholder concerns
- Separates the notion of view from *viewpoint*, where a viewpoint identifies the set of concerns and the representations/modeling techniques, etc. used to describe an architecture to address those concerns.
- It provides for capturing *rationale* and inconsistencies/unresolved issues between the views within a single architecture description.

21

21

IEEE 1471 - Conceptual Framework



22

22

Conceptual Framework - Motivation

- Establish terms and concepts for architectural thinking
- Serve as a basis for evolution of the field where little common terminology exists
- Provide a means to talk about *Architectural Descriptions* in the context of
  - System Stakeholders
  - Life Cycle
  - Uses of Architectural Description

23

23

IEEE 1471 - Viewpoints

- Views are well-formed:
  - Each view corresponds to exactly one *viewpoint*,
  - *A viewpoint is a pattern for constructing views*,
  - Viewpoints define the rules on views.
- No fixed set of viewpoints:
  - IEEE 1471 is “agnostic” about where viewpoints come from,
  - Defines templates how viewpoints are declared.
- Concerns drive viewpoint selection:
  - Each concern of a system is addressed by one (or many) architectural view(s).

24

24

IEEE 1471 - Viewpoints (cont.)

Each viewpoint is specified by:

- Viewpoint name
- The stakeholders addressed by the viewpoint
- The stakeholder concerns to be addressed by the viewpoint
- The *viewpoint "language"*, modeling technique, or analytical method used
- The source, if any, of the viewpoint (e.g., author, literature citation)

25

25

Viewpoint - Example

- Viewpoint name: *Capability*
- Stakeholders:
  - client, producers, developers, and integrators
- Concerns:
  - How is functionality packaged?
  - What interfaces are managed?
- Viewpoint language:
  - Components and their dependencies (UML component diagrams)
  - Interfaces and their attributes (UML class diagrams)
- Also known as: Static, Application, Structural viewpoints

26

26

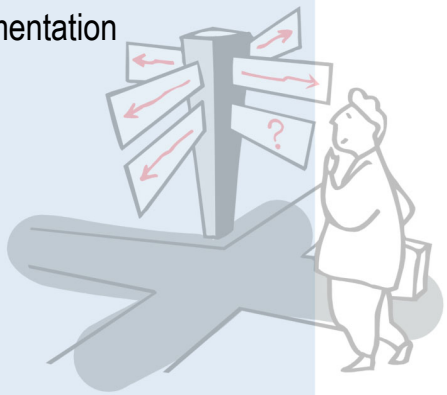
Break

27

27

Outline

- Design Documentation
  - Purpose of Documentation
  - Views
  - **Notations**
- Design Evaluation



28

28

Graphical Notations

- Popular approach to document software architectures using graphical notations
  - ☞ *"a picture can convey more than a thousand words"*
- Generally easy to understand and use
  - ☐ Tool support for selected notations available (e.g., UML)
- If not used systematically, may lead to ambiguity
- Do not lend themselves for further systematic analysis
  - ☐ Principal purpose: *communication* (not specification)

29

29

"De-facto" standard: Unified Modeling Language

Why UML?

- Reduces *risks* by documenting assumptions
  - ☐ domain models, requirements, architecture, design, implementation …
- Represents industry *standard*
  - ☐ more tool support, more people understand your diagrams, less education
- Is "reasonably" *well-defined*
  - ☐ ... although there are interpretations and dialects (and versions) …
- Is *open*
  - ☐ stereotypes, tags and constraints to extend basic constructs
  - ☐ has a *meta-meta-model* for advanced extensions

30

30

UML support: Package Diagram

Decompose system into *packages* (containing any other UML element, incl. packages)

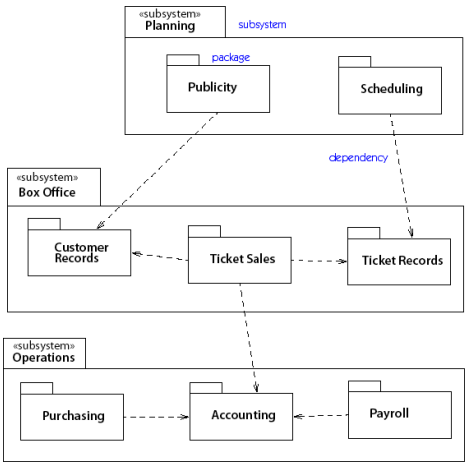


Figure 3-10. *Packages*

31

31

UML support: Deployment Diagram

Physical layout of run-time components on hardware nodes.

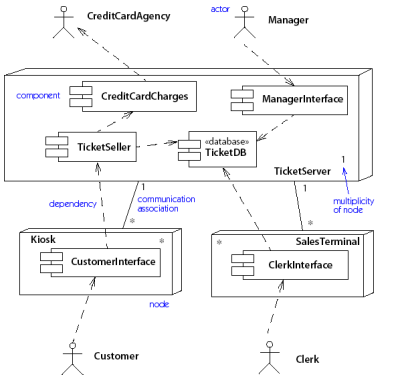


Figure 3-8. *Deployment diagram (descriptor level)*

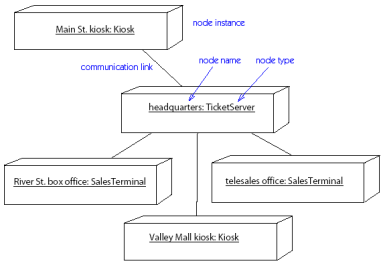


Figure 3-9. *Deployment diagram (instance level)*

32

32

Architectural Description Languages

- Can be characterized as very-high level “programming” languages
  - Exhibit many properties of a programming language; may not necessarily be computationally complete!
- Basic building blocks are high-level architectural elements:
  - Components, Connectors, and Configurations
  - (syntactical) Support for modular decomposition
- Subsume some *formal semantic model*
  - Suitable for (formal) analysis and reasoning
  - Tool support for reasoning and model checking
- Main purpose: *specification* (not communication)

33

33

Example - Darwin

```
component pipeline (int n) {
   provide input;
   require output;

   array F[n]:filter;
   forall k:0..n-1 {
         inst F[k];
         when k<n-1 bind
                  F[k].next -- F[k+1].prev;
         }
   bind
         input -- F[0].prev;
         F[n-1].next -- output;
   }
```

34

34

Additional Reading Material

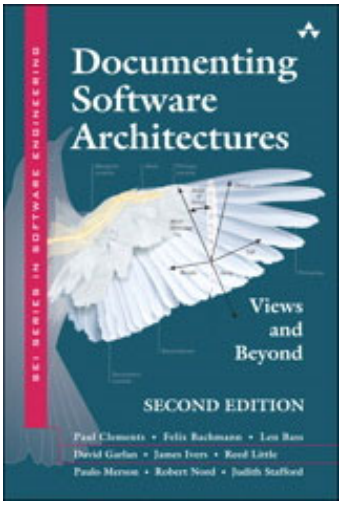
- Philippe B. Kruchten, *The 4+1 View Model of Architecture*, IEEE Software, 12(6): 42—50, November 1995.
- Dilip Soni, Robert L. Nord and Christine Hofmeister, *Software Architecture in Industrial Applications*, Proceedings ICSE '95, April 1995, pp. 196—207.
- Robert J. Allen, *A Formal Approach to Software Architecture*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, May 1997.

35

35

Documenting Software Architectures

- Second Edition of the "Documenting Software Architectures" book available.
- Draft version of the first edition available on Canvas (for your reference).

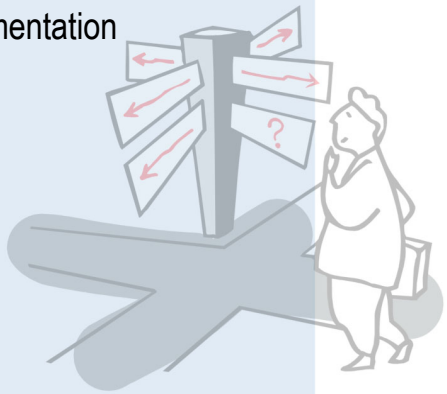


36

36

Outline

- Design Documentation
  - Purpose of Documentation
  - Views
  - Notations
- **Design Evaluation**



37

37

Why Evaluate Software Designs?

- Early detection of *design flaws*
  - ☐ Improve quality of designs
  - ☐ Opportunity to fix flaws before any code is written
- Early verification of architectural drivers
  - ☐ special focus on quality attributes
- Enhance "motivation" for improved design documentation
- (Yet another opportunity to) validate requirements

☞ *Manage both quality and risks*

38

38

Benefits

- Improved *articulation* and verification of critical architectural drivers and quality attributes
  - ☐ Identification (and resolution) of conflicting goals
  - ☞ *Improved software quality*
- Forces *clear explanation* (and documentation) of critical design issues
  - ☐ In particular at an architectural level
  - ☞ *Improved architectural rationale*
- Enhances communication between stakeholders
  - ☐ Evaluation can only happen by interacting stakeholders

39

39

… and Costs

Like any quality assurance activity, software evaluations are not for free:

- ☐ Time
- ☐ Money
- ☐ … Frustration…

40

40

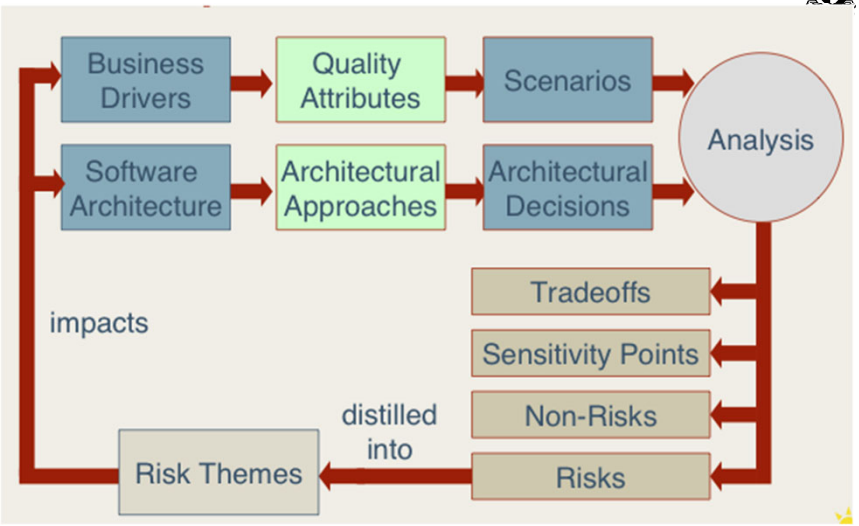
Architecture Tradeoff Analysis Method

- "Heavy-weight" method to systematically analyze software architectures
  - Comprehensive process with 5 phases/9 stages
  - Involves most stakeholders
    - ☞ *a large number of people required for a quality review!*
  - May take up several days (if not weeks)
  - Delivers comprehensive *risk analysis* of a given architectural design
- But experience shows that
  - many design deficiencies may be uncovered if all parties are *genuinely interested* in the process!

41

41

Conceptual Flow of the ATAM



42

42

Output of ATAM

- A concise *presentation* of the architecture
- Defined (and evaluated) business goals
- Quality requirements via a *collection of scenarios*
- Mapping of architectural decisions to quality requirements
- A set of *risks*, tradeoffs, and sensitivity points

43

43

Benefits of ATAM

- Gets stakeholders of project together
- Enforces critical quality goals
- Prioritization of conflicting goals
- Forces clear architectural presentation/rationale
- Improves quality of architectural documentation
- Risk identification early in the development lifecycle

Cannot hide behind "fluff"

44

44

But what about if an evaluation method such as ATAM is perceived to be too heavy-weight?

- Can we isolate and apply important concepts independent of ATAM?

45

45

"The Essentials"

- Architectural Drivers
- Prioritized Quality Attributes
  - *adapt the idea of a Quality Utility Tree*
- Scenarios
- Risk assessment

- *A more "light-weight", checklist driven approach that focuses on essential elements may suffice!*

46

46

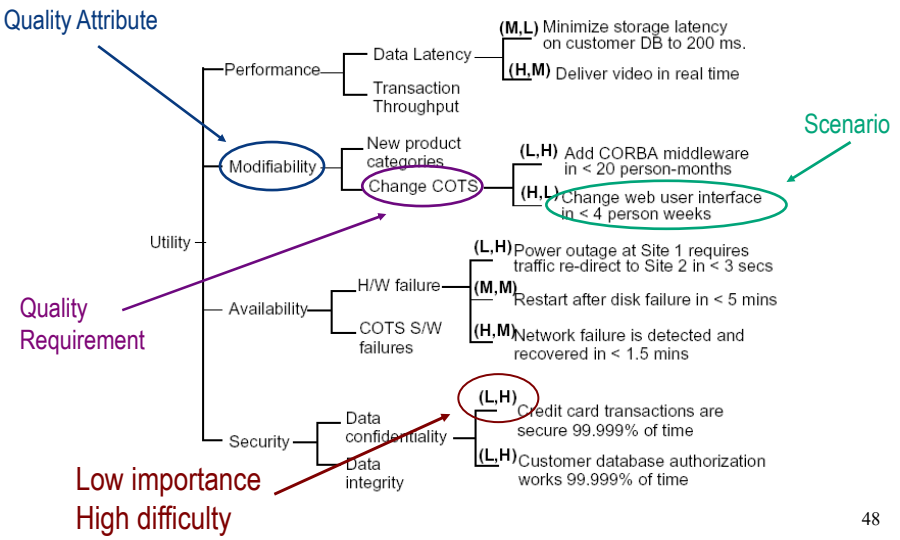
Quality Utility Tree

- Identification and prioritization of the most important quality requirements by building a *utility tree*:
  - A utility tree is a top-down vehicle for characterizing the "driving" attribute-specific requirements (i.e. architectural drivers)
  - Select the most important quality attributes to be the high-level nodes (typically performance, modifiability, security, availability etc.)
  - Add specific quality requirements at the next level
  - Scenarios are the leaves of the utility tree
  - Assess leaves based on (i) *importance* and (ii) *degree of difficulty*
    - ☞ *include skill and knowledge gaps in your assessment!*
- Results in a characterization and a *prioritization* of specific quality attribute requirements.

47

47

Quality Utility Tree - Example



48

48

A Basic Truth

Any form of organized quality assurance approach is better than none…

- *Evaluating the (architectural) design of a software system is such an organized QA approach.*

49

49

Questions for Review

1. What type(s) of View(s) would best be used to describe the various elements of the messenger system from the previous weeks question? Explain your reasoning.
2. If you maintain a popular site that cannot afford to have any downtime. Which tactic would be the most useful to stop a DDoS from halting your site?
3. What are the advantages and disadvantages of having one set of documentation for an architecture available for all stakeholders vs individually crafted sets of documentation for each group of stakeholders within a project?
4. Software Architecture in Practice (2nd Edition, Chapter 5) highlights that there are three ways to achieve the quality attribute availability. These three methods are Ping/Echo, Heartbeat and Exceptions. Discuss the advantages and disadvantages of each, and an appropriate scenario for each of them.

50

50

"Question" to Answer

*Using **six** sentences, summarize the most important/key concepts that were covered in this Unit of Study and illustrate why these concepts are important for developing (large-scale) software systems.*

------- *A self exercise (no submission required).*

51

51

Required Pre-Reading Week 12

- Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice (4th or 3rd Edition)*, Addison-Wesley, Chapter 21 (Evaluating an Architecture).

Week 12:

- Review … & Info on Final Test

52

52