TECHNISCHE UNIVERSITEIT EINDHOVEN
Faculteit Wiskunde en Informatica

TU/e

Exam 2II45 (Software Architecture) **Block B** on Wednesday 21 January 2008, 14.00h–17.00h

---

Work clearly. Read the entire exam before you start. **Motivate each answer concisely and to the point.** Maximal scores per question are given between parentheses. The maximum total score is 20 points on 11 questions.

We provide a set of hints for answering the questions. These are not the only correct answers, and in most cases they are not fully written out.

1. (2) How does architectural design relate to other software development activities?

   Hint Architectural design (AD) follows requirements gathering and analysis. AD needs (an initial set of) requirements as an input, and serves to validate and elaborate requirements. AD precedes detailed design and coding (a.k.a. construction, implemenation), and has an implemenation 'blue print' as output, which serves both as a technical road map for construction, and as a planning and progress monitoring reference. An AD description provides an early opportunity to review solution ideas. Code reviews and unit testing are based on AD module specifications. Integration testing also uses the AD as frame of reference. AD descriptions provide documentation, thereby simplifying maintenance. AD descriptions can be (partly) reused in other projects.

2. (2) When and why is it important to describe multiple architectural views?

   Hint When (this is ambiguous, since it can be interpreted as 'at what times' and 'under what circumstances'): Typically, in most cases; in particular with multiple groups of stakeholders (each group has its own viewpoint(s), served by corresponding views), and with larger software products where a single viewpoint cannot capture all relevant architectural issues in a comprehensive and accessible way.

   Why: Separation of concerns to manage complexity. Different stakeholder groups have different concerns, and different abilities to understand issues (do not bother them with irrelevant details). Different concerns often require different specialists to handle them.

3. (1) Describe a generic viewpoint that is *not* one of Kruchten's 4+1 viewpoints.

   Hint E.g.: security viewpoint, safety viewpoint, performance viewpoint, usability viewpoint, business viewpoint, ... (You should also explain what a view from such a viewpoint will offer, and why it is not subsumed in one of Kruchten's 4+1 viewpoints.)

4. (1) What usabilty aspects are an architectural concern?

   Hint Usability is not only a matter of an external user interface for human users. There are also various cross-cutting issues that concern usability, such as system-wide consistent undo and cancel operations, and internationalization; indicate why this is so.

   Designing an extensible user interface through plug-ins is also an architectural concern.

   N.B. Utility (available functionality) and performance are orthogonal to usability.

5. (2) Explain how *Module Architecture Control* can be performed.

   Hint See Lecture 7 *Module Architecture Control using Relation Algebra*. Maintain "conformance" of the "derived" module architecture (as extracted from the implementation) with the "intended" module architecture (based on documentation and information from software architects). By formalizing both the intended as well as the extracted module architecture, they can be compared. Differences require action: improvements (of either "intended", "derived", or both) or justifications (e.g. optimizations which "should" not be visible at the architectural level).

6. (2) Present a general and a specific scalability requirement in the form of a *Quality Attribute Scenario*.

   Hint Example of a general scalability quality attribute scenario:

   Source system owner

   Stimulus request to accommodate more concurrent users (usage parameter)

   Artifact the system, incl. computing platforms

   Environment normal operation, design time, run time

   Response add extra memory to servers, add extra servers (architectural parameters)

   Response measure cost of additional hardware, change in performance

   A concrete scalability quality attribute scenario:

   Source system owner

   Stimulus request to accommodate five times more concurrent users

   Artifact the main server cluster

   Environment normal operation

   Response increase the number of servers no more than sixfold, without recompiling the software

   Response measure performance as measured by average number of typical requests processed per minute may not drop more than 10%

   Also see slide 8 of ADS Architecture Lecture (Block A). Do not confuse this with a (plain) performance QAS.

7. (2) Describe the notion of *tactic* to achieve quality, and give an example of a modifiability tactic.

   Hint See slides 26–32 of Lecture 8.

   Tactic (slide 26): Design decision that influences control of a quality attribute response.

   Modifiability tactics: See slide 29. N.B. Do not confuse the goal of a tactic (e.g., "prevent ripple effects") and actual tactic(s) to accomplish this goal.

8. (2) What is the ATAM? Which ATAM activities can and cannot be used for *creating* an architecture?

   Hint ATAM stands for Architecture Trade-off Analysis Method. It is a way of organizing and carrying out a qualitative evaluation of architural designs for software-intensive systems. See ATAM slides of Lecture 9.

   Some activities of the ATAM could (should?) be carried out during architectural design: identify business drivers and architectural approaches, generate a quality attribute utility tree, brainstorm and prioritize scenarios.

The actual evaluation steps are not suitable: explain the ATAM, present the architecture, verify business case and quality attribute scenarios, analyze architectural approach against requirements.

9. (2) What is a *Component Model* and how can it play a role in architectural design?

   Hint See CBSE slide 45, Lecture 9: A component model specifies the standards and conventions that are needed to enable the composition of independently developed components.

   The architecture describes the organization of a system in terms of components (in a broader sense) and their relationships. One way to do so is to choose a component model, specify or select components for this model, thereby reusing the entire composition infrastructure and possibly also existing components. The relationships between components still need to be described.

10. (2) Compare *Object-oriented Development* and *Component-Based Development* (CBD) from an architectural perspective.

    Hint See CBSE slides 38–40 of Lecture 9: in particular, OO focuses on modularity for construction, where CBD takes modularity further, viz. independent design, production, deployment, and use. As a consequence, CBD (as opposed to OO) involves "really" late binding, explicit dependencies, and binary reuse.

    From an architectural perspective this means that a component-based design offers better modularity, hence better modifiability and more opportunity for reuse. By reusing well-tested components, the quality can also be better guaranteed. But this comes at a price: more generic interfaces and a more expensive implementation effort (because it needs to take care of operation in a more general—i.e., less controllable—environment).

11. (2) "Dynamic aspects of the software cannot be analysed without running it." Would you agree with this statement? If yes, explain why. If no, provide a counterexample.

    Hint See slides on Reverse Engineering of Lecture 10. N.B. The negation of the statement is not: "*All* dynamic aspects can be analysed without running it", and also not "Some dynamic aspects cannot be analysed by running it".

    The best case is to be made for "Yes, some dynamic aspects can be analysed without running it". Appropriate RE tools can extract behavioral models (e.g., sequence diagrams, activity diagrams, state diagrams, Petri nets) from source code through static analysis, i.e., without executing it.

    In fact, one can even defend "Yes, some dynamic aspects can only be analysed rigorously without running it", because by running the software you can explore only a very limited part of the (often very big) state space, whereas formal analysis techniques (such as model checking) can verify properties for all reachable states.