

Foundations of Software Architecture
Subject Guide

Authors

John Reekie, University of Technology, Sydney

Contributors and reviewers

Zenon Chaczko, University of Technology, Sydney

Lian Loke, University of Technology, Sydney

Terms of Use

This document is Copyright © H. John Reekie 2004–2005. It is distributed under the terms of the Creative Commons **Attribution-ShareAlike 2.0** license. See below.



Attribution-ShareAlike 2.0

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:



Attribution. You must give the original author credit.



Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Creative Commons
<http://www.creativecommons.org>

The legal code of this license
<http://creativecommons.org/licenses/by-sa/2.0/legalcode/>

Contents

1 Overview of the subject	1
1.1 Learning objectives	1
1.2 Quality attributes of this subject	1
2 Face-to-face activities	2
2.1 Teaching schedule	2
2.2 Attendance	2
2.3 Instructor communication	2
2.4 Timing and scheduling	3
3 Content delivery modules	4
4 Assessment	6
4.1 Assignments	6
4.2 Examination	6
4.3 Bonus mark	6
4.4 Minimum requirements	6
4.5 Misadventure	6
5 Assignments	7
6 Examinations	8
6.1 Exam structure	8
6.2 How to do well in the exams	8
7 Bonus mark	9
7.1 Keep a blog	9
7.2 Write a research paper	9
8 Subject resources	10
8.1 Subject-specific resources	10
8.2 Additional reading	10
A How to do well in this subject	11
B How to use a discussion board	12
C How to give a good presentation	13
D Quality attributes for education	14

1 Overview of the subject

Software architecture is the study and design of the most pervasive aspects of a software system, and in particular, of complex software systems. As such, it takes on the task of the high-level decomposition of the system into components, the relations and interaction between them, and how, when combined, they form a system that meets the needs of the system's many stakeholders.

In this subject, you will be introduced to the field of software architecture from a pragmatic and practice-based viewpoint. The subject assumes that you have a good grounding in the basics of programming and software construction, and a good understanding of the basic principles of software engineering and software lifecycles. This subject aims to extend your thinking about software systems to larger and more complex systems.

1.1 Learning objectives

After completing this subject, you will be able to:

- Understand and describe a software system in terms of its stakeholders, system interfaces, architectural qualities, and business requirements.
- Develop and refine multiple views of a software system architecture, based on the conceptual, execution, and implementation architecture.
- Understand the key issues in implementing complex distributed or real-time systems, including networking, distribution, performance, testing and reliability, and real-time qualities.
- Implement architectural prototypes using off-the-shelf components, middleware, and custom code.

1.2 Quality attributes of this subject

This subject was explicitly designed with several *quality attributes* in mind:

- **Relevance:** the content of the subject is relevant to both the workplace and the University.
- **Fairness:** the subject delivery and assessment is fair and reasonable.
- **Enjoyability:** the subject is enjoyable both to take and to deliver.

You will have an opportunity to tell us how well you think this subject has met these quality attributes in a survey at the end of the semester. For more information on quality attributes, see Appendix D.

2 Face-to-face activities

This section gives you all the information you need to know about when and where to show up for all subject-related activities. It also specifies due dates and times for all assessable deliverables.

2.1 Teaching schedule

There are thirteen coursework semesters. Of these, one is dedicated to student presentations and twelve are allocated to coursework delivery. Of the twelve, some may be used for activities such as guest lectures, coverage of additional material, and revision.

Face-to-face teaching time in the subject consists of:

- Lectures. Lectures run for one hour. Please make sure that your mobile phone is turned off, and do not use a laptop computer during lectures.
- Laboratory and tutorial sessions. These sessions build on the material in the lectures, and run for two hours. You are expected to attend the full lab or tute session.
- LDC time. LDC consultation times will be posted early in the semester. Priority will be given to teams over individuals. Additional LDC time may be scheduled by making a request on the discussion board.

2.2 Attendance

The scheduled activities for this subject are three hours per week.

You are required to attend all scheduled class sessions.

That includes tutorials and laboratories as well as lectures. The regular class session is how the instructors disseminate course materials, subject content, returned assignments, and study advice. This is not a correspondence course, and you will find it difficult to achieve a good grade if you do not attend all classes.

Lack of attendance at any scheduled milestone demonstration will result in a zero mark for that milestone.

2.3 Instructor communication

The instructors in this subject are trying to deliver you the best learning experience possible with a very constrained budget. It is important that you help yourself and others by following these guidelines:

1. Use the discussion board as the first and primary means of contact with the lecturer (outside of scheduled class times).
2. Email is only to be used for issues that cannot be handled in class or on the discussion board.
3. All email must have the string “[48433]” (including the square brackets) in the subject or it will be ignored.

2.4 Timing and scheduling

Table 1 gives the schedule for the whole semester. Note that you are expected to attend all scheduled class sessions:

- Lectures are scheduled from 1 pm to 2 pm on Mondays.
- Labs and tutorials are scheduled from 2 pm to 4 pm on Mondays (that is, immediately following the lecture).

Table 1 also gives the dates when assignment deliverables are due. Deliverables must be submitted as follows:

- Written deliverables must be placed on the UTSONline dropbox by **noon** of the specified date. Written deliverables must be a *single uncompressed PDF file*. The file name must contain your team number and the milestone number.

Any submissions—zip files, Word documents, strange file names, and so on—that do not meet these simple criteria *will not be marked*.

If you have hand-written or hand-drawn elements in your deliverable, hardcopy of those parts *only* are to be submitted at the start of the lecture on the specified date. These will not be returned.

- Programming deliverables are to be checked into the CVS or subversion repository by **noon** of the specified date.

Marked copies of each written deliverable will be returned, wherever possible, one week after submission.

Week		Date	Lecture	Activities	Deliverables
Cal	Sem				
31	1	5 th August	1	Tute 1	—
32	2	12 th August	2	Tute 2	—
33	3	19 th August	3	Lab 1	—
34	4	26 th August	4	Lab 2	—
35	5	2 nd September	5	M1 demo	Milestone 1
36	6	9 th September	6	Lab 3	—
37	7	16 th September	Mid-term	M2 working session	—
38	8	23 rd September	—Tutorial week—		
39	—	30 th September	—VC week—		
40	9	7 th October	7	M2 demo	Milestone 2
41	10	14 th October	8	Lab 4	—
42	11	21 st October	9	Lab 5	—
43	12	28 th October	A	Lab A	—
44	13	4 th November	10	M3 demo	M3 demo
45	14	11 th November	Presentations	Presentations	Milestone 3
46–48	—	14 th Nov – 2 nd Dec	—Final exam—		

Table 1: Subject schedule, Autumn 2005

3 Content delivery modules

There are a total of twelve delivery modules, each consisting of a lecture, readings, and other learning resources. The first six are considered to be “foundation” modules, and the remaining modules explore specific topics in more depth.

Module 1: Software Architecture in Context

This is the introductory module to this subject. It explains the concept of software architecture and places architectural work into the context in which software system development takes place.

Module 2: Conceptual Architecture

This module introduces the first effort at decomposing a system into an architecture, based on functional requirements. Simple tools like usage narratives and use-case maps are used to support the derivation of the initial conceptual architecture.

Module 3: Execution Architecture

The execution architecture expresses the system in terms of concurrent elements and hardware elements. This module explains execution architecture, and examines concurrent subsystems, concurrent objects, and deployment.

Module 4: Implementation Architecture

The “build-time” view of the system is contained in the implementation architecture. Program elements and modules, and off-the-shelf components and infrastructure form the third leg of the “architecture triad.”

Module 5: Architectural Styles

Experience and practice have led to a number of recurring architectural structures. These structures (styles or patterns) are part of the “body of knowledge” of the field of software architecture.

Module 6: Quality Attributes

This module takes a deeper look at quality attributes, and how they are analyzed and represented in the three architectures.

Module 7: Realtime Systems Architecture

In realtime systems, *time* is the dominant force that shapes the architecture. Specific architectural responses such as time-based tiers and stream processing are covered.

Module 8: Web Architecture

HTTP front ends are almost mandatory on client-server systems these days. This module provides an accessible introduction to the basic elements of web architecture. It builds on the N-tier architectural style, and covers both the principles of web systems and the implementation architecture of web servers and web services.

Module 9: Component Frameworks

Component-based software engineering (CBSE) is an active area of work in the effort to improve the way in which we build complex systems. This module surveys the current state of the art and practice and explains how components, infrastructure, and middleware support complex software systems.

Module 10: Design and Culture

Design is one of the fundamental activities of software development. This module explores the nature of design as relevant to software architecture, and is a journey through design patterns, iteration and exploration, and the importance of culture and the tension with process.

Guest Lecturer modules

These modules are delivered by industry-practitioner guest lecturers.

Module A: Computer-Integrated Manufacturing

Computer-Integrated Manufacturing is a case study that illustrates how many of the concepts covered so far come together in large, complex, “real-world” systems.

Module B: Enterprise Systems

Enterprise systems architecture spans the technology and systems that support the running of a business enterprise. This lecture examines this level of architecture and discusses techniques used to analyze and visualize enterprise systems.

4 Assessment

Figure 1 summarizes the assessment structure of this subject. The assessment is split between teamwork-based assignments and individual examinations. All teamwork submissions also include an individual component.

4.1 Assignments

There are two assignments, delivered in a series of parallel milestones spread roughly evenly throughout the semester. The assignments are worth 30 marks each. See Section 5 and the Assignment Guide for more information.

Assignment marks will be scaled at the instructor’s discretion, if there is not reasonable participation by all team members. The individual reflections, electronic logbooks, and the source code checkin logs will be examined to verify reasonable participation.

4.2 Examination

There are two examinations. The mid-term exam is a one-hour exam, worth ten marks, and is held during a scheduled class session. The final examination is held during the University’s examination period, and is worth 30 marks.

4.3 Bonus mark

Five bonus marks can be earned for extra work; see Section 7.

4.4 Minimum requirements

In order to pass this subject, you must achieve a total mark for all assessment items of 50 or more.

4.5 Misadventure

If there is a legitimate reason why your assessment was or will be adversely affected, you must submit an *Advice of Scheduling Difficulty* or *Request for Special Consideration* form to the UPO, together with supporting documentation. Requests for special consideration from students that have not attended a majority of class sessions, will be denied.

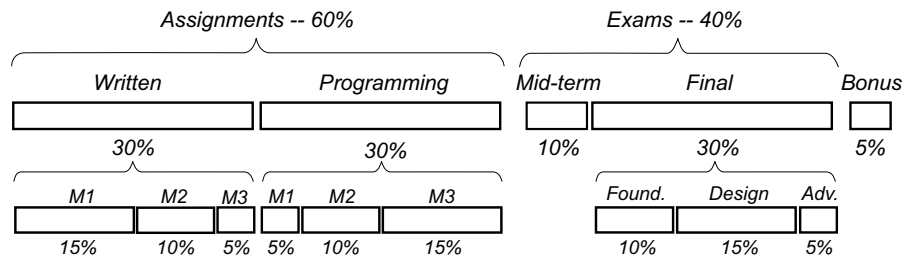


Figure 1: Assessment structure

5 Assignments

For the purposes of this undergraduate class in software architecture, the delivery structure of the assignments mimics the first two phases of the Unified Process, which are called Inception and Elaboration. The Unified Process emphasizes the importance of producing an early working version of the system in the form of an executable architectural prototype. In parallel with this work, the architectural design is carried out, resulting in an artifact called the Software Architecture Document. Therefore, you will be working on two assignments throughout the semester:

- The written assignment produces the Software Architecture Document.
- The programming assignment produces the Executable Prototype.

Both are delivered incrementally, in a series of three milestones. Figure 2 summarizes the structure of the assignments. The marks for each assignment change with each milestone, as follows:

	Milestone 1	Milestone 2	Milestone 3
Written	15	10	5
Programming	5	10	15

Each milestone must be delivered on the specified date and time. There will be **no extensions**. It is therefore vital that you deliver on time, and if you cannot deliver what you would like on time, then just deliver less. (Use Google to find articles on the software management approach called *time-boxing*.)

Each assignment has a page limit. (Cover pages and appendixes are not included in the page limit, but will not be assessed.) This is to encourage you to write more clearly and succinctly, and also to make your work easier to mark. Marks will be deducted for each page over these limits:

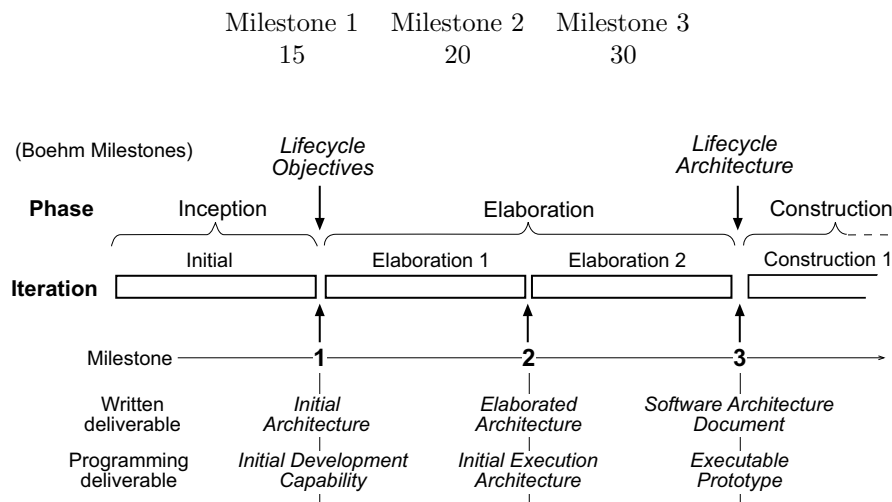


Figure 2: Assignment milestones

6 Examinations

Exams in any subject are the cause of significant angst for many students. Exams are intended to sample your knowledge of the subject and provide an individual assessment component (as opposed to the teamwork-based assignments). This section describes the structure and style of the exams, and gives some tips on how to do well in them.

6.1 Exam structure

In this subject, exams consist of one of two types of questions:

1. Short-answer questions

Short-answer questions are similar to the questions in the Exploration Packs. They are meant to explore your understanding of the general concepts of Software Architecture. The questions are chosen from all material covered in lectures, tutorials, labs, and assignments.

The content of the supplementary readings is not included in short-answer questions (unless it has specifically been covered in class).

2. Design questions

Design questions exercise your understanding of Software Architecture as gained by doing the assignments and labs. The questions are typically framed to provide you with some elements of an architectural design, and ask you to analyze and further develop this design.

There are typically two groups of design questions. The first group are all required, and cover roughly the same ground covered in the assignments, labs, and exploration packs. The second group of questions covers more difficult material, typically introduced towards the end of the semester. You will have a choice of which questions out of these to do.

6.2 How to do well in the exams

Because of the structure of the exam and the style of questioning, most students will find it quite difficult to do well by “cramming” for the exam. The best way to gain the knowledge that the exam asks for is to participate fully in all class activities. That is:

- Attend and participate in all class sessions
- Work through the questions and exercises in the Exploration Packs
- Participate in the online discussions and activities
- Work closely with your teammates on all assignments

To repeat: memorizing lecture notes or readings is not a particularly effective strategy for doing well in the exam. Note also that the lecture slides are a *guide* to the topic only, and not the only examinable material.

7 Bonus mark

Five bonus marks can be awarded for work over and above the normal subject requirements. Following are the default ways in which you can receive this mark. Instructors may also choose to allow other means of being awarded the bonus mark. You can only get the bonus mark once.

Bonus marks are awarded on a go/no-go basis—that is, you either get 5 or zero.

7.1 Keep a blog

Keep a blog (“weblog”) of your experiences in and thoughts on this subject. The blog can be viewed as an electronic equivalent of the reflective portion of the learning journal that you may have kept in other subjects. You will need to use the blogspace provided by the instructor, unless you can make a good case that there is a good reason to use your own. Please note that there is no such thing as a “private” blog: blogs are published on the web and are therefore readable by anybody.

Blog entries are expected to be thoughtful and reflective. That is, they examine something that you learnt and how you went about learning it. They can also contain your own original ideas related to software architecture. A simple summary of a lecture or tutorial is not adequate to receive the bonus marks for blogging.

A blog entry is expected approximately once a week. Retrospective entries created three weeks later do not count! (Blog entries are time-stamped.) The final blog entry is to be completed by the end of the last week of class.

To be approved for this option, you must create your first blog entry by the end of semester week 3, and email the instructor that you wish to be considered for the bonus mark for blogging. There is no guarantee that your blog will be accepted—if you are not absolutely confident that you will meet the criteria, you should do this well in advance of the deadline, so that you can respond to the instructor’s feedback.

7.2 Write a research paper

Write a short (6 to 8 pages) research paper on some aspect of software architecture. The paper is expected to include a survey of relevant papers in the field, and add some original content. To be approved for this option, submit an extended abstract (one page), which includes at least four references, to the instructor by the end of semester week 7.

The final version is due at the end of the last week of class. You will be required to submit a PDF version of your paper for inclusion in the “subject showcase.”

8 Subject resources

8.1 Subject-specific resources

These are the resources available for this subject.

- Subject Guide. This document.
- Software Architecture Workbook. Describes all tutorials, assignments, and laboratory exercises, and includes week-by-week exercises.
- Excerpts from *Introduction to Software Architecture*. Required readings written specifically for this class.
- Software Architecture Reader. Sections of various books and papers that supplement the material provided in the above.

8.2 Additional reading

The following books (from which many of the readings in the Reader are taken) are all in the UTS library. You may wish to use these to further your knowledge of the field.

- *Design and Use of Software Architectures*, by Jan Bosch. Addison-Wesley, 2000.
A nice introduction to the “industrial” view of software architecture.
- *Software Architecture in Practice, Second Edition*, by Len Bass, Paul Clements, and Rick Kazman. Addison-Wesley, 2003.
Useful supplementary reading for students that want to read a lot.
- *Applied Software Architecture*, by Christine Hofmeister, Robert Nord and Dilip Soni. Addison-Wesley, 1999.
Interesting look at factors and system context.
- *Pattern-Oriented Software Architecture—A System of Patterns*, by Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. John Wiley and Sons, 1996.
Contains a good collection of the basic architectural patterns.
- *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* by William J. Brown, Raphael C. Malveau, Hays W. “Skip” McCormick and Thomas J. Mowbray. John Wiley and Sons, 1998.
Excellent. Contains patterns at all levels, not just architecture.
- *Documenting Software Architecture: Views and Beyond* by Paul Clements and other authors. Addison-Wesley, 2002.
A slightly different set of viewtypes than in this subject.

A How to do well in this subject

In this section, we give some advice that you should think about before and while studying this subject.

Resources

In case you've never thought of it this way, here's something to think about:

Studying a subject is a game of maximizing a result given limited resources.

In other words, you are trying to maximize your marks in this subject, but you only have limited resources to do so. One of those limited resources is your own time. That's clear enough. But the other limited resource that you have is *access to the instructor*.

You shouldn't waste either of these resources. Use them wisely.

Class and lab attendance

In case it's not obvious already, the best thing you can do to increase your mark in this subject is to show up at every class. The instructors use the face-to-face class time to *deliver* the subject, and if you're not there, it won't be delivered to *you*. You will find that it's quite difficult to recover from the downward pressure that lack of attendance will place on your marks.

A similar comment applies to lab attendance. The scheduled lab session is really your best opportunity to ask instructors about the lab exercises and programming assignments, so you'll be a lot better off showing up and doing the exercises than not.

Spread time evenly

The subject has been designed so that work *can* be spread fairly evenly throughout the semester. Therefore, it's a good move to spread time evenly through the semester, instead of using "lurch mode" (spending all your time just before the assignments are due and the final exam).

Study the current topic now

Many students postpone studying the topic of the current week until some later week. This tends not to work very well in this subject, because by then the class has moved on and the instructor is focussed on talking about the current topic. So, you should do the exploration packs in the week they are presented, and discuss the topics on the discussion board.

Note: it's a particularly bad idea to leave the exploration packs until the exam study period. By then, the instructor's brain has basically packed up and gone on vacation, so you'll be on your own if you wait until then.

B How to use a discussion board

The discussion board is a valuable resource, but—like most things—it only works properly if you use it properly. In particular, you will need to learn how to ask questions properly:

- A focussed question will generally get a better answer than an open question.
- You learn as much from answering a question as you do by asking it.

Remember that this subject is not about rote learning, but about principles and their application. You should therefore see the board as a way to build up a shared knowledge amongst the members of the class, rather than as a question-and-answer type of resource. When asking a question, try and explain what it is you *do* understand as well as asking what you don't, as this helps to frame your question. Try answering questions from others, even if you're not sure you have the right answer. Often, you will increase your understanding of the subject matter even more.¹

Finally, it's a good idea, if an instructor responds to a post of yours, to respond again with additional thoughts of your own. This lets the instructor know that you are thinking about the material, and the probability that he or she will respond to future posts of yours will increase.

¹It is often said that the best way to learn something is to try and teach it.

C How to give a good presentation

Most people under-estimate how much work it takes to develop an effective presentation, and very few people give one. It's actually very difficult. We are not expecting a world-class presentation, but we would like to see a good one, and so the following recommendations are intended to help you maximize your marks for this part of the assessment.

1. Plan your slide presentation early. Use a pencil and paper to sketch out the flow of the presentation, well before starting on creating the actual slides.
2. Aim to convey a few important pieces of information, rather than as much information as possible. You have only 15 minutes, and three points conveyed effectively will deliver more value to the audience than a dozen points conveyed poorly.
3. When you present, don't act like you're in front of a camera for the first time. Stand up, don't fidget, divide your attention between the slides and the audience, and speak clearly. And don't talk to your laptop!
4. Rehearse and *time* your presentation. Unless you've given a lot of presentations before, you will have no idea how long it is going to take. Most likely, you will have too many slides. Remember, you will be stopped when you get to 15 minutes, so make sure you can say what you want to say within that time.
5. Finally, get to your scheduled presentation session well in advance of the scheduled start time, so you can get your presentation onto the computer and check that everything is working OK. Note: you should not rely on a USB memory stick working with a computer you have never used before.

D Quality attributes for education

As you will learn in this subject, a software architect explicitly considers the *quality attributes* of a system as well as the *functional* needs. The quality attributes of a software or computer system includes things like performance, reliability, maintainability, and so on.

So, when designing this subject, I asked myself what the quality attributes of a tertiary subject might look like. After some thought, I came up with the following desirable quality attributes:

- Relevance

Whatever the subject teaches, it should be *relevant* to the current practice of software development, both in the workplace and in the rest of the student's undergraduate work.

- Fairness

The subject should be *fair*. That is, the workload needed to pass this subject is reasonable for almost all students.² In addition, assessment is reasonable and is based on the complete context in which assessment occurs—the time available to perform the task, the content covered in lectures and labs, and so on.

- Enjoyability

The subject should be *enjoyable*. That is, the students should find the lectures reasonably interesting, and the material both accessible and challenging. The instructors are also entitled to enjoy delivering the subject, after all it's enough work preparing it so we may as well have some fun delivering it.

- Survivability

This subject must exhibit *survivability*. For a tertiary subject, Survivability is the ability of a subject to survive the removal of its creator from its delivery. To this end, the subject has been placed on the web as “open courseware.”³

²The University defines a reasonable workload for a 6 credit point subject to be 150 hours over the whole semester, or about nine hours per week, so you should plan to spend that much time per week on this subject.

³See <http://creativecommons.org/>.