Integrating Software Architecture Topics into a Software Engineering Curriculum

John Grundy

Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

As Software Architecture issues become more important for Software Engineers, the importance of including Software Architecture-related topics in a Software Engineering curriculum also increases. This paper reports on the author's experiences in integrating various Software Architecture topics into a Software Engineering programme, focusing in particular on four courses the author has designed and run over the past four years. An example Software Engineering programme is outlined, then for each selected course the Software Architecture-related topics, teaching style and tools used by the students are discussed. Important lessons learned about Software Architecture education from the development of these courses are summarised, along with plans for future curriculum evolution. It is hoped that this report on our underlying philosophy, approaches and experiences with integrating Software Architecture topics into a curriculum will assist others embarking on a similar mission.

Key words: Software Architecture, Software Engineering Education, Teaching Approaches, Teaching Tools

1. Introduction

As software systems continue to become more complex, and the use of distributed, heterogeneous systems increases, the need for good knowledge and understanding of Software Architectures by Software Engineers also increases. Software Architecture is principally concerned with how developers organise the software components of a system [1, 18]. Software engineers need to concern themselves with how different parts of a system are divided into manageable pieces ("components"), how these components are inter-related, trade-offs between different approaches to component division, inter-relationship and communication support, integrating heterogeneous components into a system, modelling and analysing the Software Architecture of systems, including performance and robustness, and generating designs and implementations based on Software Architecture models. In recent years the range of potential Software Architectures and related implementation technologies for a system have increased drammatically with the advent of componentware systems [12, 19], middleware technologies [11, 17], E-commerce systems [10], Electronic Data Interchange [2], and decentralised systems [3]. In addition, there has been a lot of research into Architecture Description Languages [1], CASE tool support for Software Architecture modelling [7, 8], and an increasing emphasis on architecture and systems evaluation [14].

Many Software Engineering curriculums have been developed, tried and documented in recent years [6, 13, 15, 20]. Most emphasise important topics like Project Management and Software Processes, Requirements Engineering, Object-oriented Analysis and Design, software reuse, testing and quality assurance, human-computer interaction issues, and systems maintenance. Few currently address Software Architecture-related topics in any depth, if at all, although some discuss related topics of Design Patterns, Middleware technologies and briefly address issues concerned with Architecture Modelling [6, 13]. This lack of Software Architecture topics is perhaps not so surprising when one considers that Software Architecture, as an important area of Software Engineering in its own right, has not been identified as such until quite recently [18]. An interesting comparison can be drawn with Computer Systems Engineering programmes, where the study of Computer Systems Architecture, i.e. the organisation of computer hardware and networks, is fundamental to the whole programme [9, 16]. Typically many courses and parts of courses address topics like hardware organisation, networking approaches, hardware and network modelling, and performance analysis.

The need to introduce topics covering Software Architecture theory and practical skills, and to give students a range of experiences in modelling, analysing, developing and evaluating various Software Architecture approaches, seems clear, given the increasing importance of Software Architecture-related issues for practicing Software Engineers [1]. But how does one go about integrating Software Architecture-related topics into a Software Engineering programme? This paper discusses the experiences of the author in trying to do this over the past four years at the University of Waikato in New Zealand. In particular, it focuses on four courses that the author has been responsible for designing and discusses how Software Architecture topics have been integrated into these courses. Some courses have had such material "added" to replace or augment existing material, while others have been designed with Software Architecture as the keystone topic, used to underly all facets of the course material and teaching style. Positive and negative experiences to date are discussed, along with plans for future curriculum evolution including Software Architecture topics as part of a Bachelor of Engineering in Software Engineering degree programme.

Section 2 gives an example Software Engineering programme used as the framework within which courses containing Software Architecture topics have been developed and run. Section 3 describes a core Year 2 course which was modified to include an introduction to some basic Software Architecture issues and practical techniques. Section 4 discusses a Year 3 Software Engineering Project course which was significantly modified to emphasise Software Architecture issues, and to give large project experience in which students model and develop distributed Software Architectures. Section 5 discusses two Year 4 (i.e. honours year) courses, one which was augmented with some Software architecture material, and one which was designed from the beginning with Software Architecture as the keystone topic on which the course material and project experience focuses. Section 6 discusses positive and negative experiences to date with designing and running these courses, and outlines some future curriculum development plans based on these experiences.

2. An Example Software Engineering Programme

The Department of Computer Science at the University of Waikato has offered a popular Software Engineering programme as part of its 4 year Bachelor of Computing and Mathematical Sciences (BCMS) degree for many years [5, 6]. Figure 1 illustrates the main courses in this curriculum. Courses in itallics are optional, and courses in bold are the ones designed and run by the author, and are discussed in the following sections.

Key places where topics relating to Software Architecture could well be envisioned in this curriculum have been marked with an asterix (*), with exact topics dependent on other course material, prerequisite structures and the course lecturers' interests and expertise. Such topics might include the concept of software organisation vs hardware organisation (Year 1, 105 course), database server concepts and usage (Year 2, 219 course), client-server concepts and practical techniques (Year 2, 209 course), different approaches to software component organisation (Year 3, 301 course), architecture modelling, analysis and practical experience (Year 3, 314 course), advanced database server architectures and algorithms (Year 3 and 4, 319 and 481 courses), formal specification and analysis of Software Architectures (Year 4, 424 course), Software Architectures for GUI and groupware systems (Year 4, 425 course) and various topics relating to Software Architecture theory, practice and experience (Year 4, 482 course).

Unfortunately it turned out that little emphasis had originally been placed on Software Architecture-related topics in this curriculum. There are various reasons for this, chief amoung them the lack of understanding and experience of Software Architecture issues by the faculty and the amount of other material deemed necessary to cover. The author has been responsible for the design, running and evolution of four courses in this curriculum (209, 314, 425 and 482), highlighted in bold in Figure 1. One of my key aims over the past few years has been to address the lack of Software Architecture topics in the curriculum as a whole and in these courses in particular.

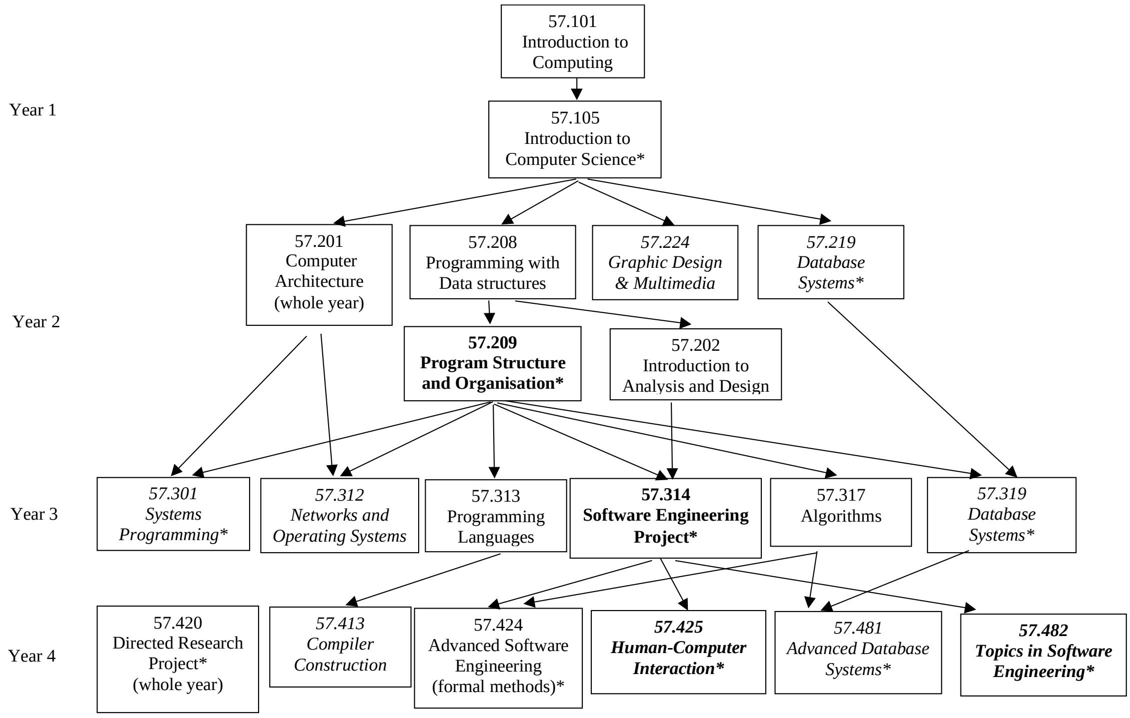

Figure 1. An overview of the BCMS(Software Engineering) Programme courses.

In the Year 2 Program Structure and Organisation course ("209"), Software Architecture as a concept was introduced and students shown various motivations for more complex software architectures, mainly the need to distribute and share data and processing. Basic pratical design and implementation techniques and project experience using client-server architectures focusing on database server usage was introduced. In the Year 3 Software Engineering Project course ("314"), a complex Distributed Information Systems project had to be built, motivating the need for an understanding of basic Software Architecture theory and practical modelling and implementation techniques. Students develop an appropriate Software Architecture for their system and debate this amoung themselves and with the course lecturer. They also implement a prototype of this system's architecture and do basic testing of it. At Year 2, limited tools and conventional teaching techniques were used, while at Year 3 CASE tool support and project-based learning formed the main approach to supporting teaching of Software Architecture material.

At Year 4, two divergent approaches were taken to introducing Software Architecture topics. In the Human-Computer Interaction course ("425"), Software Architectures for User Interface development, particularly the development of groupware systems was added to the User Interface Development half of the course the author taught. For the Topics in Software Engineering course ("482"), Software Architectures formed the keystone on which the course was built i.e. Software Architectures was the key focus which underlay and motivated other topics. A variety of quite novel teaching approaches were used at Year 4 level to support teaching of Software Architecture material, including simulations of architectures, middleware technologies and groupware systems, various practical design and programming experiments, and large project-based learning experiences. The following sections examine the motivation for the course content, teaching style and tools used in these four courses in more detail.

3. Year II: Program Structure and Organisation

This Year 2 course (referred to from here by its numerical label, 209) is a required core course for all Computer Science students, including those taking the Software Engineering programme [6]. Thus material relating to Software Engineering needs to be carefully chosen and integrated, as students from all BCMS programmes must take and pass this course. The main rationale for 209 is to provide students with practical techniques and experience in developing "larger" programs than in earlier prerequisite courses. Thus techniques to support design and testing of programs, as well as technologies for building "larger" programs, are introduced.

There was almost no Software Architecture material in the original 209 course design. This greatly restricted the "practical" kinds of systems students would encounter in the course, basically restricting them to single user, monolithic programs. The author modified the 209 course material to include file servers and database servers, allowing students to gain experience in developing systems with shared data and multiple users. The rationale for this limited enhancement was to build upon existing student skill sets and course material, while allowing the usage of commonly encountered client-server software architecture models to be experienced by students.

Figure 2 outlines the main topics, teaching methods and tools used in the revised 209 curriculum. Basic theoretical concepts about client-server architectures are introduced, along with basic technologies used to realise such systems. These are motivated by the limitations of programs that use monolithic, single-user supporting architectures. Students do not build servers or low-level client-server communication software, but are limited to using provided servers via well-defined APIs. This was a carefully considered course design decision that allows students to gain experience designing and building simple client-server-based systems, but hides low-level networking and server management issues. Basic testing techniques for client-server systems are also introduced and used. The addition of these basic Software Architecture topics to 209 can be seen as a way to expose students to common client-server architecture concepts and focus on giving them experience using such modelling and implementation techniques to develop realistic systems, without getting bogged down in low-level communication and server-side processing detail.

Course Content	Teaching Methods	Tools Used
1. Text and binary file I/O 2. Structured Program Design 3. Structured Programming 4. Object-oriented Program Design 5. Object-oriented Programming 6. GUI Development 7. Databases 8. Database API Programming 9. Intro to Software Engineering	• Lectures, mainly on practical techniques • Example programs and designs • Individual "projects" (i.e. large assignments) • Tests on theory/practice	• C++ • mSQL database server & API • Haskell • Word processor (for designs, documentation)

Figure 2. Program Structure and Organisation course content, teaching methods and tools.

It can be argued that experience developing servers and communication software would be realistic for Year 2 students in their fourth programming course, perhaps with an introduction to different Software Architecture models and middleware technologies, such as RMI or CORBA. Unfortunately this does not suit 209 due to the amount of other course material, and the fact that most students will spend two more years doing their BCMS degree, and thus Year 3 and 4 courses can cover this material in much more depth. This "black-box" approach to servers and client-server communication technologies was thought to better suit this course's primary aim of providing students with "programming-in-the-medium" practical techniques and experience, as more system design, client implementation and testing material can be included.

The teaching style adopted by 209 involves providing basic theoretical grounding of client-server Software Architectures in lectures, with emphasis on the object-oriented design and implementation techniques used to model and build client-side processing and user interface programs. Example models and programs are provided which are adapted by students to build their own clients. A relational database SQL client is used to create database tables and a reflection mechanism object data files for server-side data management. Two practical projects out of four provide students experience in using client-server architecture modelling, design, implementation and testing skills. Tests and a final exam include mainly practical-related questions about students knowledge of modelling, implementing and testing parts of client-server systems. This fairly conventional teaching style allows students to be presented with basic theory, a range of practical techniques in a "cook-book" style, and good experience in using the techniques on realistic system development.

We do not use any CASE or Software Architecture modelling tools in 209, and client-server software development is done in C++ with provided server programs and server API interfaces. CASE tool support was found to hinder rather than support the learning of basic architecture and program design concepts. Thus we focused on teaching students basic client-server modelling and implementation techniques without powerful tool support, emphasising basic conceptual and practical skill development rather than learning a particular tool or tool set's capabilities and idiosyncrasies.

4. Year III: Software Engineering Project

This Year 3 course, referred to from here on by its numerical designation 314, is the key Year 3 required course of the BCMS Software Engineering programme [6]. It is also taken by students in some other BCMS programmes to expose them to group work and large project development. The author designed and evolved this course over several years with the primary aims of giving students experience in working in groups, experience in working on a large, carefully scoped Software Engineering project, and providing them with a range of practical skills for engineering software systems. Theoretical concepts in Software Engineering are de-emphasised and left for Year 4 courses to cover in depth. Figure 3 outlines the main course content, teaching methods and tools used for 314.

Course Content	Teaching Methods	Tools Used
1. Project management 2. Requirements Engineering 3. Object-oriented analysis 4. Software Architecture 5. Object-oriented design 6. Implementation 7. Testing and Quality Assurance 8. Maintenance 9. CASE Tools 10. Professional Practice	• Group project for whole course • Group meetings (some with lecturer) • Lectures, mainly focusing on basic theory and practical techniques/case studies • Case study materials • Project work (done in group) • Project reports, oral presentations	• Rational Rose™ for analysis, design, some software architecture modelling • Choice of C++, Java, Delphi™ with basic socket comms APIs • MS Word™ for documentation, some software architecture modelling • PowerPoint™ for presentations

Figure 3. Software Engineering Project course content, teaching methods and tools.

Earlier versions of 314 focused on conventional client-server systems development, typically simple Information Systems applications. The course covered more theoretical aspects of software engineering, limited practical techniques and experience with assignments, and included little group work. Very little complex Software Architecture modelling and implementation was done, and no analysis or maintenance. It was clear such an approach was highly unsuitable for educating Software Engineers in modern distributed systems development.

Besides a major reorganisation of 314 course content and teaching methods, Software Architecture developed over the last three years as a key topic, bridging the gap between OO Analysis and Design, with Software Architecture-related issues coming to underly much of the course material. Students are provided with a variety of different Software Architecture styles, some simple Architecture Description Languages are introduced and used, and techniques discussed for determining appropriate architectures based on OOA objects and non-functional requirements. OO Design uses the Software Architecture determined for a system to motivate service object choices, help determine inter-object calling protocols and help determine necessary design-level communication and multi-user support functions. Architectures are prototyped and resultant code tested systematically. Experience in maintaining an existing implementation due to architecture (and requirements) changes is given. Students thus gain a good appreciation of different architecture styles, gain experience in using basic architecture modelling techniques, and gain experience in implementing and maintaining reasonably complex architectures with socket-based communication technology and database servers.

A key to ensuring students gain practical skills in modelling, designing and implementing sophisticated software architectures is the focus of the course around group project work. The project used for the past three years is a swipe card entry system – a system with interesting user interface, distributed system, security, performance and robustness characteristics. Students must develop their own requirements and specification for this system, and determine an appropriate software architecture which satisfies the specification. They need to refine the specification into a design, constrained by the software architecture model they have developed, and implement a prototype of the system. An interesting consequence of the group work aspect of the course is how much debate goes on between group members when trying to determine a suitable architecture to meet their specification's needs. This has usually been the area where most discussion and alternative models have been produced, and is often one of the more interesting periods of discussion of work with the course lecturer. Students appear to learn a great deal about different architecture models and designs through this process, despite the fact that only one architecture and corresponding object-oriented design is necessary in their project report. An additional important characteristic is the prototyping of the system, including a suitable architecture for the prototype. This is often no where near as sophisticated as the students' "full system" architecture model (often containing multiple servers, caches, multiple network links and so on!), but is sophisticated enough that a challenging implementation is developed.

Students use the Rational Rose™ CASE tool for their object-oriented analysis and design modelling, and some use UML deployment and component diagrams to help describe their architecture. Many, however, use a more rich architecture description language introduced in lectures but which has no existing tool support, as UML's support for software architecture modelling is relatively poor. A key finding by students is that detailed discussions of architectures using English are as important if not more so than graphical diagrams describing them. Comparing and contrasting alternative Software Architecture models for a system was also found to be an important part of a project report. To date 314 has used socket-based Java, Delphi and C++ APIs to support system prototyping, along with relational database servers. More sophisticated distributed systems middlewar, such as RMI, DCOM or CORBA, have not been included due to time constraints, but may be used in future if the basics of such technologies can be covered in preceding courses.

5. Year IV: Human-Computer Interaction and Topics in Software Engineering

The Year 4 Human-Computer Interaction course, designated 425, is an option for Software Engineering programme students, and has a major component focusing on user interface development. As a consequence of the increasing development of distributed systems, e-commerce systems, groupware applications and novel user interface systems (including virtual reality and wearable computers), knowledge of software architectures has become essential for user interface developers. This is particularly the case with respect to distributed and groupware applications [4]. Figure 4 outlines the course content, teaching approaches and tools used for 425.

Course Content	Teaching Methods	Tools Used
1. Human factors 2. Psychology of everyday things 3. User modelling 4. User interface design 5. User interface implementation 6. Groupware systems 7. Novel user interface technologies 8. User interface evaluation	• Lectures, mainly on HCI/UI theory • Readings on theory/practice • Student presentations • Practical programming assignments • Activities/tutorials • Individual project for whole course	• Tcl/tk • Java (AWT) • GUI design tools • MS Word™ for documentation, designs • PowerPoint™ for presentations

Figure 4. Human-Computer Interaction course content, teaching methods and tools.

Software Architecture-related topics were included in the user interface implementation and groupware parts of 425 to ensure students had some knowledge of such issues relating to complex user interface software development. When developing web-based applications, distributed systems user interfaces and groupware applications, quite sophisticated software architectures are often necessary in order for such systems to be effectively realised. Topics in 425 relating to this thus include an overview of user interface design for client-server and other distributed software architectures, architectures for groupware systems to support shared workspaces, asynchronous editing and group awareness, and wearable computer architectures.

Practical experience in prototyping some such systems is given using Java in small assignments. Experience in developing large systems with such architectures was not given in the project work for 425, with emphasis on user interface design, interface prototyping and interface evaluation, as these were originally seen as the more important issues for the project work. Instead, tutorial activities relating to distributed system interfaces and groupware interfaces used simulation of such systems, including their interface behaviour, architectures and inter-process communication. Examples included having students simulate groupware application interfaces by writing on the same piece of paper simultaneously ("shared workspace") or merging parts of a document from different sheets of paper ("asynchronous editing"). Architecture issues were then explored i.e. what ways could such software "lock" areas of work, transmit information about who was editing what, and coordinate work. Simulations of network traffic between multiple users editors highlighted the complexity of the architectures of such software.

Tool support for user interface design was provided in 425, but not for architecture modelling. Implementation of prototype distributed system interfaces and groupware tools used Java, and included a distributed address book and collaborative drawing editor. Simple Java database server and socket-based communications for these distributed applications were used. In retrospect, the lack of large project experience with distributed system user interfaces for students has been a failing of 425.

The Year 4 course Topics in Software Engineering, numbered 482, until 1999 focused on theoretical systems analysis and data modelling. The author completely redesigned this course from scratch, using many of the same design principles and teaching methods devised for 425. However, 482 was designed with Software Architecture being the keystone topic in Software Engineering around which the rest of the course was organised. Figure 5 outlines the main topics covered in 482, its teaching methods and the tools used for project work.

Course Content	Teaching Methods	Tools Used
1. Software Processes 2. E-commerce systems 3. Requirements Engineering 4. Object-oriented Analysis 5. Software Architecture 6. Object-oriented Design 7. User interface Design 8. Middleware technologies 9. CASE and metaCASE tools 10. System Evaluation	• Lectures, mainly focusing on basic theory and practical techniques • Case study materials • Readings, mainly theory and practice • Student presentations • Practical design and programming assignments • Activities/tutorials, mainly simulations and workshops • Individual project for whole course	• Rational Rose™ for analysis, design, some software architecture modelling • Java with socket, RMI and CORBA comms • SQL Server™ database server • MS Access™ database client • MS Word™ for documentation, some software architecture modelling • PowerPoint for presentations

Figure 5. Topics in Software Engineering course content, teaching methods and tools.

E-commerce systems development was chosen as a representative, modern distributed system for which Software Engineering techniques are needed to help develop. These systems are by nature distributed, may exhibit groupware facilities, are often heterogeneous and have several interesting user interface, security, robustness, performance and inter-operability characteristics that need to be addressed. Requirements engineering and object-oriented analysis for such systems is thus rather more complex than for many other kinds of systems. Similarly, the range of potential Software Architectures for such systems is large, with many trade-offs between different options possible. 482 requires students to investigate various Software Architecture issues for such systems, including modelling and analysing potential architectures, developing suitable object-oriented designs realising such architectures, use a variety of middleware technologies and user interface design techniques to implement such systems, and evaluate, among other things, the performance of the architectures for such systems.

A mix of lectures, readings, student presentations, practical assignments, tutorial activities and a large e-commerce system development project are used in 482. Readings, lectures and student presentations give theoretical background and an introduction to practical development techniques. Assignments gave students opportunities to design and build prototype distributed systems using a variety of techniques, using Java, SQL Server™, Java sockets and RMI, and basic CORBA services. Tutorial activities combined design workshops and architecture and middleware simulations to provide an interactive forum for exploring distributed system development issues. Example simulations included comparing and contrasting socket, database server, RMI and CORBA middleware architectures by groups of students "acting out" the performance of such technologies for an e-commerce application. Client-server, multi server, point-to-point and decentralised architecture styles were simulated. CASE tools for distributed systems modelling and analysis were designed and discussed, and commercial distributed systems evaluated, including their architectures and architecture performance characteristics. The realistic e-commerce system development project reinforced for students the key role Software Architecture-related decisions have during modern distributed systems development. The project also gave students the opportunity to use architecture modelling, design, implementation and evaluation techniques "for real".

Tools used in 482 included Rational Rose™, Java, Java RMI and CORBA IDL compilers, SQL Server™, and MS Access™. Students found using MS Word™ for most of the architecture modelling and documentation more effective than Rose. The lack of an integrated environment supporting architecture modelling and refinement to OO design, user interface, server and middleware implementation, and testing of different parts of a system, was a major problem. This resulted in lower than hoped for productivity and some confusion when different parts of a system were modified, especially changes to the software architecture model being used. No Software Architecture analysis tools were available, which students thought would assist them in developing more suitable architectures for their systems.

6. Discussion

In general we have had very positive experiences in integrating Software Architecture topics into the four courses outlined above. Software Architecture has become a foundational area of practice in Software Engineering and thus must become an important area of study in any Software Engineering curriculum. An introduction to Software Architecture basic theory and practice in our Year 3 Software Engineering Project course has given students an opportunity to not only gain experience in reasoning about and developing distributed system architectures, but also in debating trade-offs between different approaches among themselves. Software Architecture as the underlying foundation for the Year 4 Topics in Software Engineering course has been validated by the excellent course reviews of students, the range of sophisticated e-commerce prototype systems developed using various architectural approaches, and the quality of architecture modelling and documentation in student project reports. Some of the most positive outcomes were from the practical assignments and interactive tutorials focusing on architecture modelling, simulation, experimental prototyping and evaluation. Students clearly gained an excellent appreciation of the key theoretical concepts and practical skills needed in Software Architecture from these activities. The addition of Software Architecture material to our Year 4 Human-Computer Interaction course complemented the user interface development material, and gave an improved understanding of complex groupware and distributed system interface architectures and the abilility to design and build these.

The introduction of simple client-server and database server architectural abstractions in our Year 2 Program Structure and Organisation course allows students to design and build more sophisticated applications, while minimising the need for them to use complex communication technologies and implement server-side processing. This has generally worked quite well, although we feel that experience in designing and building some simple server and communications software in this course would be useful for the following Year 3 courses to build upon. Currently we are limited in the material about Software Architectures we can introduce in the Year 3 Software Engineering Project course due to the quite limited introduction of this topic in the Year 2 course. A more detailed coverage of Software Architecture topics in Year 3 would also allow students to gain some experience in modelling, designing and implementing a variety of different architectures, rather than only realising one architecture in their project. This is particularly important for those students not taking the Year 4 courses which provide more detailed architecture investigation. Requirements Engineering issues related to Software Architecture are poorly addressed throughout this current curriculum, even at Year 4 level, due to time constraints. Similarly, while some architecture analysis and evaluation is conducted in the Year 4 Topics in Software Engineering course, this ideally would be more comprehensive.

As part of the Bachelor of Engineering in Software Engineering development at the University of Auckland, some enhancements to the preceding curriculum can be envisaged to better address such issues. A greater emphasis on software architecture, particularly client-server development, is currently being undertaken in the equivalent Year 2 course Software Design and Construction. This could be built upon in Year 3 courses, one similar to the Software Engineering Project course outlined above and one focusing on Software Architecture-related topics. The project course could continue to ensure students put Software Architecture practical skills to work on a large, group project, while a Software Architecture course can cover a much wider range of theoretical and practical modelling, analysis and implementation technologies. Included in this course would be more emphasis on developing requirements for complex distributed systems and evaluating the architectures of such systems (robustness, performance, security, integrity etc). Other potential Year 3 courses on User Interface development, Engineering (System) Design, networking and Software Engineering concepts would all include Software Architecture-related topics appropriate to the main focus of their course material. Similarly, Year 4 courses focusing on large, industry-based project experience and advanced Software Engineering topics would include a mix of Software Architecture theory, practice and experience.

7. Conclusions

Fostering a good understanding of Software Architecture theory, teaching practical modelling, analysis, implementation and evaluation skills, and giving experience in developing a range of software architectures for complex, distributed systems are key aims of the course developments described in this paper. Software Architecture topics need to be integrated into a modern Software Engineering curriculum at various levels, ranging from introductory concepts and basic practical skills in early years to detailed investigation of sophisticated architectures, modelling techniques, validation and analysis approaches, implementation technologies and evaluation methodologies in later years. We have found that a limited introduction of basic software architecture concepts and practical experience in developing simple client-server applications works well for laying a foundation for later courses to build upon. Project-based courses which require complex, distributed software architecture solutions provide students with a forum for discussion and debate over possible solutions, a clear motivation for developing a suitable architecture in the context of a realistic project, and experience in prototyping and evaluating such an architecture. Other Year 3 and Year 4 courses can be enhanced by introducing Software Architecture-related topics, particularly where existing course material needs to draw upon such topics due to recent technology and design practice changes. Courses dedicated to Software Architecture concepts, practice and experience are suitable at Years 3 and 4 in a Software Engineering curriculum and we believe are becoming essential components of Software Engineering programmes, due to the increasing importance of this area.

Acknowledgements

Course content and curriculum discussions with Steve Jones, Mark Utting, Steve Reeves, Mark Hall, Rick Mugridge, John Hosking, Neville Churcher, Andy Cockburn, Geoff Roy, Matt Humphrey, and Bill Rogers have influenced course design decisions at various times. The support of Professor Mark Apperley for Software Engineering curriculum development at the University of Waikato over many years has been most appreciated.

References

1. Bass, L., Clements, P. and Kazman, R. Software Architecture in Practice, Addison-Wesley, 1998.
2. Crowley, R.T. Edi Charting a Course to the Future : A Guide to Understanding and Using Electronic Data Interchange, Research Triangle Consultants, 1993.
3. Gokkoca, E., Altinel, M., Cingil, I., Tatbul, E.N., Koksal, P., and Dogac, A., “Design and implementation of a Disrtributed Workflow Enactment Service,” in *Proceedings of 2nd IFCIS Conference on Cooperative Information Systems*, Charleston, SC, USA, June 1997.
4. Greenberg, S. Teaching Human Computer Interaction to Programmers, *ACM Interactions*, Vol. 3, No. 4, July-August 1996, ACM Press, pp. 62-76.
5. Grundy, J.C. Experiences with Facilitating Student Learning in a Group Information Systems Project Course. In *1996 Software Engineering: Education and Practice Conference*, pages 12-19, IEEE CS Press, Dunedin, New Zealand, January 24-27 1996.
6. Grundy, J.C. A Comparative Analysis of Design Principles for Project-based IT Courses, In Proceedings of the 2nd Australian Conference on Computer Science Education, Melbourne, July 2-4 1997, ACM Press.
7. Grundy, J.C., Hosking, J.G., Mugridge, W.B. and Apperley, M.D. Tool Integration, Collaboration and User Interaction Issues in Component-based Software Architectures, Proceedings of TOOLS Pacific’98, Melbourne, Australia, Nov 24-26 1998, IEEE CS Press, pp. 299-312.
8. Liu, A. Dynamic Distributed Software Architecture Design with PARSE-DAT, In Proceedings of the 1st Australasian Workshop on Software Architectures, Melbourne, Australia, Nov 24 1998, Monash University Press.
9. Kendall, E.A. A Pattern Based Curriculum for Computer Systems Engineering, *10th Conference on Software Engineering Education and Training*, April 13-16 1997, Virginia Beach, VA, IEEE CS Press.
10. McClaren, C.H. E-Commerce: Business on the Internet, South-Western Publishing Company, 1999.
11. Mowbray, T.J., Ruh, W.A. *Inside Corba: Distributed Object Standards and Applications*, Addison-Wesley, 1997.
12. O'Neil, J. and Schildt, H. *Java Beans Programming from the Ground Up*, Osborne McGraw-Hill, 1998.
13. M.J. Oudshoorn, A.L. Brown and K.J. Maciunas. Simulating Real-Life Software Engineering Situations in the Classroom. In *1996 Software Engineering: Education and Practice Conference*, pages 20-25, IEEE CS Press, Dunedin, New Zealand, January 24-27 1996.
14. Richardson, D.J. and Wolf, A.L. Software testing at the architecture level, In Proceedings of the 2nd International Workshop on Software Architecture, San Francisco, October 1996, ACM Press, pp. 68-71.
15. Roy, G. and Verarrt, V. Software Engineering Education: from an Engineering Perspective. In *1996 Software Engineering: Education and Practice Conference*, pages 256-262, IEEE CS Press, Dunedin, New Zealand, January 24-27 1996.
16. Sage, A. P., Systems Engineering and Information Technology: Catalysts for Total Quality in Industry and Education, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol 22., No. 5, 1992, pp. 833-864.
17. Sessions, R. *COM and DCOM: Microsoft’s vision for distributed objects*, John Wiley & Sons 1998.
18. Shaw, M. and Garlan, D. Software Architecture : Perspectives on an Emerging Discipline, Prentice Hall, 1996.
19. Szyperski, C.A. *Component Software: Beyond Object-oriented Programming*, Addison-Wesley, 1997.
20. V. Veraart and S. Wright. Experience with a Process-driven Approach to Software Engineering. In *1996 Software Engineering: Education and Practice Conference*, pages 406-413, IEEE CS Press, Dunedin, New Zealand, January 24-27 1996.