Theory and Practice of Tertiary Science and Engineering Education

Arnold Pears,
Uppsala University, Sweden
Arnold.Pears@it.uu.se

Stephen B. Seidman,
Texas State University, TX, USA
seidman@txstate.edu

October 16, 2017

Contents

Part I

Theory and Methods

Chapter 1

The Tertiary Teaching and Learning Landscape

This handbook is written for university teachers of science, technology engineering and mathematics (STEM) disciplines who are seeking to place their teaching within a context of pedagogical research. Even after decades of research on tertiary-level teaching and learning in many disciplines, the results of this work have only just begun to have a broad impact on university teaching practice as part of an holistic effort to enhance Scholarship of Teaching and Learning. The purpose of this book is to bring aspects of this research tradition to STEM educators in universities, and to present practical examples of its application in the university classroom. The book provides an overview of the teaching and learning research that fuels this change in the working environment. We also discuss the characteristics of a scholarly approach to academic leadership at levels ranging from individual courses to degree programmes and curriculum design.

In this chapter we provide an overview of the changing priorities in modern higher education, helping to contextualise scholarly teaching practice in the broader context of the development of the academy, and the associated academic traditions, roles and responsibilities, as they are reflected in the higher education of the 21st century.

Traditionally, access to tertiary education has defined and served a privileged segment of a nation's population. Holders of university degrees have enjoyed higher salaries as well as status and influence in government and industry, establishing an intellectual elite that has blossomed in many countries in the years following the Second World War. Policy makers, observing the higher salaries and adaptability of university graduates, have enacted legislation to increase participation in higher education for their citizens in the belief that this will result in increased prosperity for individuals and nations.

This is only true if nations can use their citizens' education and skills to establish themselves as knowledge economies. This can be done by creating high-quality educational systems that can produce the knowledge workers required by a knowledge economy. The value of education therefore lies in its quality and the availability of jobs that require high level education.

Several specific challenges for higher education arise in the context of establishing a knowledge society. First there is a shift away from providing a research education to a minority, to providing broader education in the service of industry, government and society. Second, issues of diversity and equity of access to higher education become increasingly important [1]. To some extent these initiatives have been criticised as leading to inflation of qualifications and establishment of a plethora of tertiary qualifications in areas that are not traditionally academic. We can conclude that, together with an expansion of the academic sector through the establishment of new universities, these factors have created significant challenges for higher education institutions of the 21st century.

The teaching and learning challenges inherent in the expansion of the higher education sector and concomitant increase in access to tertiary education for citizens has been addressed in several studies of higher education [2,3]. Learners have increasingly disparate backgrounds with the associated variations in prior knowledge and experience, not to mention issues of age, language, learning culture and family expectations [4].

Increasing awareness of these challenges during the decades since the publication of Boyer's book "The Priorities of the Professoriate" and works which developed that position in regard to the practice of higher education [5, 6] in 1990 have seen a greatly increased realisation of the critical importance of educational aspects of academic life. As a result, there has been increased attention paid to the need to enhance capacity for scholarship in teaching and learning among academics. Boyer's description of a scholarship specifically dealing with teaching and inspiring learners to engage in the discipline sparked a world- wide interest in enhancing higher education quality through staff development, and staff involvement in applied education theory across all disciplines.

The higher education research literature has engaged vigorously with this situation from a pedagogical perspective [7–9], providing a range of models and approaches. There is also a considerable literature in disciplinary education research which explores the implications of the broader scholarly findings for teaching and learning in Physics, Chemistry, Computer Science and Engineering.

What does this mean for us as teachers? Faced with expectations to engage learners and to practice scholarship of teaching and learning it can be hard to know what is expected, or how to engage meaningfully in scholarly based teaching practice in higher education.

The aim of the remainder of this book is to familiarise readers with salient aspects of the Scholarship of Teaching and Learning literature. Building on this we develop a hands-on framework for research informed instructional design. Finally, different applications of the framework are described using case studies in different disciplines.

Chapter 2

The Nature of Learning in Science and Engineering

Education at research universities is characterized by teaching that is informed by disciplinary research activities. At such universities, the content of faculty research permeates the educational environment. In this context, high-quality learning depends on research in both the scientific subject matter itself and in the learning of advanced disciplinary concepts.

The motivation to change the teaching and learning climate of the academy builds on a broad body of work on teaching and learning theory, quality management of organisations, and arguments regarding the nature of academic excellence.

However, educational research is very often both unfamiliar and relatively inaccessible to researchers in STEM disciplines. This presents challenges in terms of bringing new educational models to bear on teaching in these disciplines. This chapter will attempt to bridge the gap between STEM researchers and education researchers.

Several different approaches to learning with implications for knowledge retention can be traced to the infliuential research of Swedish educationalists on surface and deep learning [10]. Building on this early work of Marton, Säljö, Booth and others, Prosser and Trigwell address how the instructional environment can facilitate acquisition of knowledge in an increasingly diverse educational ecology. They argue [11, page 408] that there is a fundamental qualitative difference between a student-centric and teacher-centric view of the learning process. Specifically, they claim that a student-centered approach to facilitating learning focuses on the nature of the learning itself, placing the main emphasis on changing student conceptions of the subject matter. In contrast, a teacher-centric approach may be characterised by a focus on issues related to subject matter content and delivery. These claims are supported by [12,13]. In a more specific disciplinary context, the existence of these qualitative differences in teacher perceptions in the general population of computer science academics is supported by data collected by Pears et al. [14]

As part of this discourse on the nature of higher education, the goals of higher education have come to be defined by many researchers as developmental and student-centric [15–18].

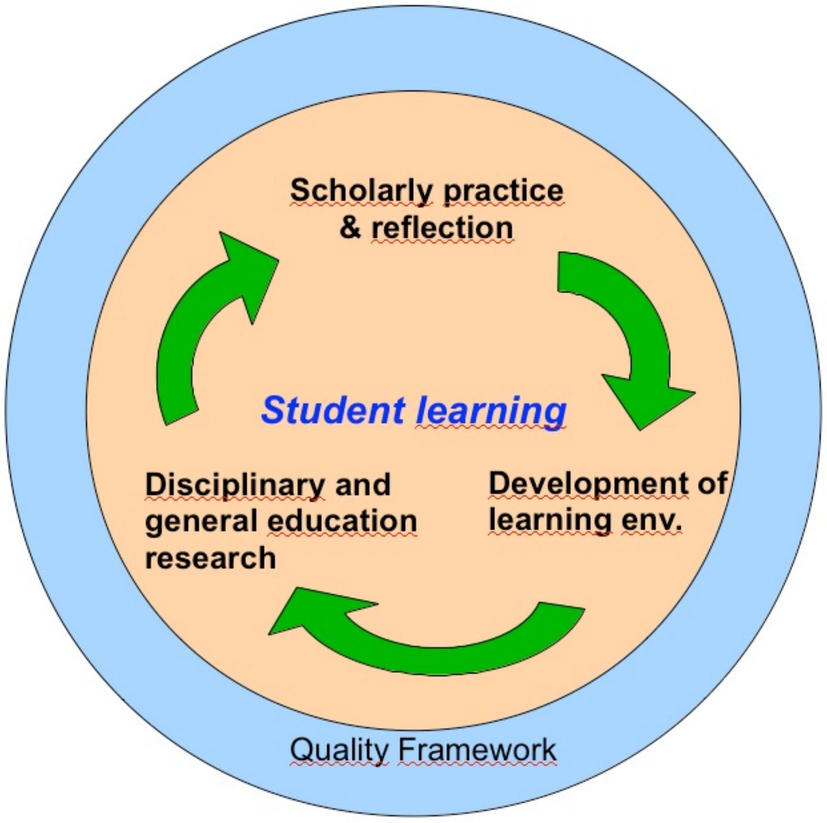
Scholarly practice & reflection
Student learning
Disciplinary and general education research
Development of learning env.
Quality Framework

Figure 2.1: Factors influencing educational excellence

2.1 Approaches to Learning

How students approach learning is dependent on the learning environment. In particular, there is a relationship between what teachers do and how students learn; this relationship has explored by a number of researchers [8, 19]. This implies that teaching would be improved if teachers were more aware of what actions they can take to structure the learning environment in order to achieve the learning outcomes they desire.

But, if empowerment in, and commitment to, scholarly teaching and learning practices is vital for the renewal of higher education, how is this to be achieved?

As we have noted, educational research results are often not easily accessible to many science researchers. Educational theories are often abstract and results couched in general terms, which presents challenges in terms of adapting new educational models to teaching in specific STEM disciplines. Likewise, educational researchers often have little exposure to advanced research concepts in STEM disciplines, which affects their ability to discern and study learning phenomena related to tertiary STEM teaching and

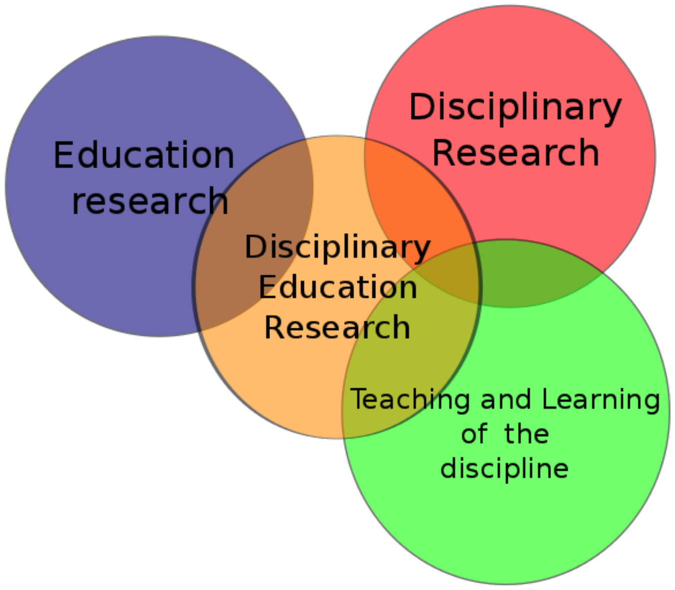
Education research
Disciplinary Research
Disciplinary Education Research
Teaching and Learning of the discipline

Figure 2.2: Role of educational research in the disciplines

It is useful to relate the goals of learning activity to to models of learner development and of the learning process. Figure2.3 drawn from Entwistle [20] integrates two perspectives on the learning process and relates them to the development of understanding and identity.

In this context, teachers' ability to facilitate student development is enhanced by a student-centric approach that incorporates conceptual change. This viewpoint is supported by research of Kember [21], who studies the impact of student attitudes and expectations on the nature and outcomes of the teaching and learning process.

Figure 2.2 illustrates the role of disciplinary education research in the complex relationship between research in the discipline, teaching of the discipline, and research in education and higher education. Disciplinary education research

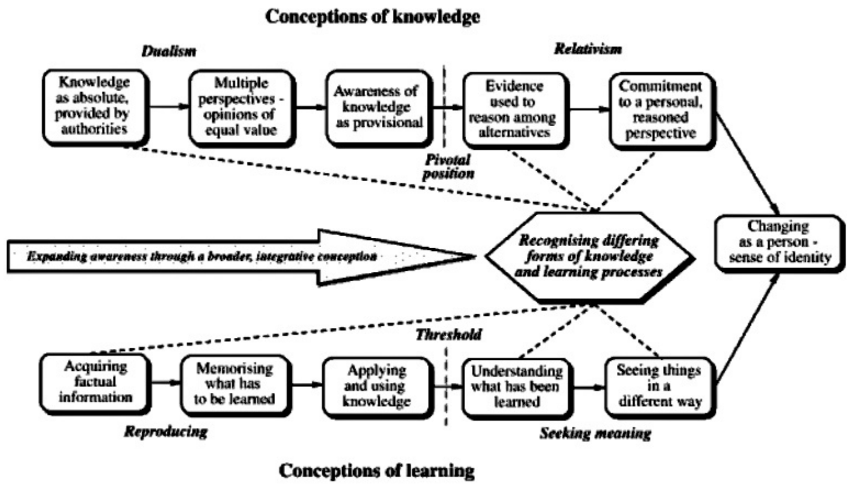
Conceptions of knowledge
Dualism
Relativism
Knowledge as absolute, provided by authorities
Multiple perspectives - opinions of equal value
Awareness of knowledge as provisional
Pivotal position
Evidence used to reason among alternatives
Commitment to a personal, reasoned perspective
Expanding awareness through a broader, integrative conception
Recognising differing forms of knowledge and learning processes
Changing as a person - sense of identity
Threshold
Acquiring factual information
Memorising what has to be learned
Applying and using knowledge
Understanding what has been learned
Seeing things in a different way
Reproducing
Seeking meaning
Conceptions of learning

Figure 2.3: Entwistle's integrative model of learner development

in higher education necessarily draws upon both knowledge of the discipline itself as well as relevant theory from research in higher education.

Research that informs higher education practice can be broadly characterised based on the particular point of departure. In much higher education research, the point of departure is a philosophy of education or a research paradigm. Taking a philosophical standpoint from the outset leads to the development of schools of educational research; studies in these traditions tend to have a common underlying epistemology. This implies that the methodology and results are rather similar in character and follow well established traditions.

In contrast, the point of departure for disciplinary education research is usually situated in the instructional context. It addresses perceived shortcomings in the instructional design, achieved learning outcomes, or student capabilities and competencies. Such research focuses on pragmatic disciplinary goals, where subject knowledge provides crucial insights as to the relative value of types of student understanding and conceptual development. In this type of research a single investigator might draw on methods from either qualitative or quantitative research, or both, in the search for enlightenment.

The eclectic nature of research practice is partially illustrated in figure 2.4

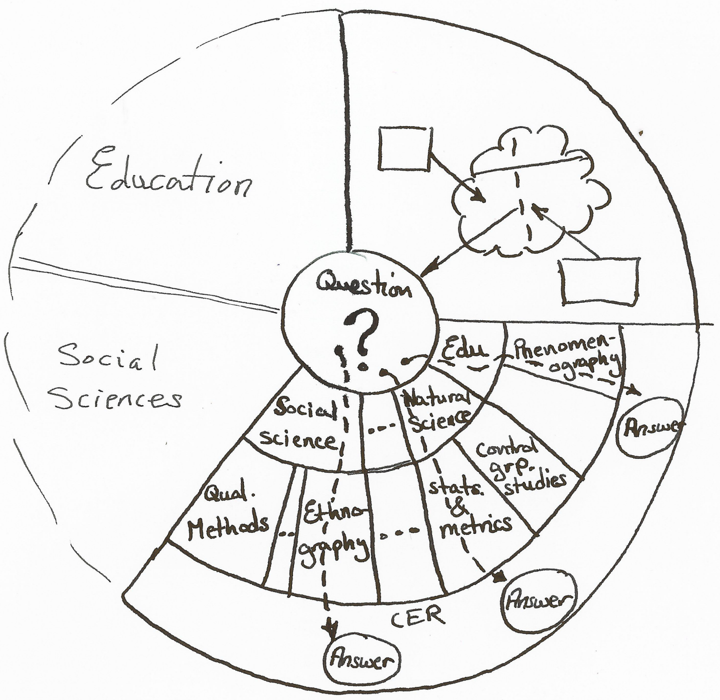

Figure 2.4: Role of educational research in the disciplines

Chapter 3

Practical Scholarship of Teaching and Learning

What is scholarly teaching practice? Boyer defined the scholarship of teaching as activity which,

> "educates and entices future scholars by communicating the beauty and enlightenment at the heart of significant knowledge."

Trigwell et al. [22, page 156] cite Ramsden's definition of teaching "The aim of teaching is simple: it is to make student learning possible." and then define the purpose of scholarly teaching as "making transparent how we have made learning possible.".

However, these definitions are abstract and the nature of scholarly teaching in higher education remains diverse and hard to capture since it can be practiced in a variety of different ways. For instance, there is a difference between implementing curricula and teaching methods based on prior research, evaluating curricula and teaching methods in the local educational context, and conducting research which aims to extend the bounds of what is known about how to teach in a discipline. In this chapter the main focus is on what Shulman [23] termed "*what we know about learning*, scholarly inquiry into how students 'make meaning' out of what the teacher says and does."

To provide a framework for the context of scholarly engagement in teaching and learning in higher education we focus on how academics can contribute to an enhanced learning environment in their discipline. Trigwell [22] in his interviews with hugher education academics identifies several intentions and strategies associated with scholarship of teaching and learning. Broadly stated one can collapse these categories into the three classes shown in figure 3.1.

Improve practice: through awareness and application of relevant scholarship in their educational practice. The aim is to maintain awareness of relevant disciplinary research literature in order to stay abreast of best practice in the area of the discipline where one is engaged in research and teaching.

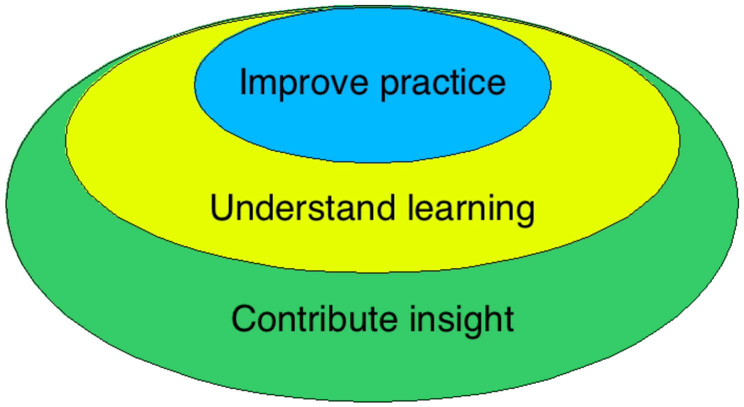

Figure 3.1: Approaches to Scholarly Teaching and Learning Practice

Understand learning: by investigating learning with the intent to understand the implications of educational innovation. This implies taking a step beyond the intention to improve practice by also collecting data and conducting systematic analysis to evaluate the effects of changes to curricula and methods of instruction.

Contribute insight: through pursuing a research vision that contributes to establishing theory and contributing evidence to a wider discourse on teaching and learning of the discipline. This involves combining a deep udnerstanding of the discipline with research in theory and practice of higher education in order to improve understanding of how students develop and become experts in the discipline.

The first two levels deal with the application of research to practice. The aim is not to do original research in teaching and learning theory. Activity focuses on applying known results and evaluating the impact of established promising practices. It seems likely that engagement at this first level will become essential for all academics in the near future[1].

Research into the nature of learning in the discipline is the focus of people who fall into the final category. This will naturally be the province of those who choose to pursue a research career in teaching and learning theory in their discipline. Research of this type is needed in order to address new educational challenges at undergraduate and postgraduate level, but cannot be expected of everyone.

On the other hand, the demands of accreditation and quality assurance agencies on higher education are best met when academic staff are operating at the second level or above. At level two instructional design is well informed and

[1]This is already implicit in the quality assurance schemes of many european countries. Look at policies formulated by the European Quality Agency for Higher Education to support this statement

motivated by research, and the outcomes are well documented through collection and analysis of appropriate data collected in the learning environment during the course. These data can take many forms and be analysed using a large number of methods in order to reason about the advantages and disadvantages of that approach to supporting student learning.

3.1 Improving Practice

Educational quality assurance in higher education is based on a continuous improvement model. Figure 3.3 provides an overview of the major activities each improvement cycle contains. Implementing quality assurance in a higher education setting is challenging. The context of learning is a complex one, and highly dependent on context and the participants. Background knowledge and experience among staff and students vary from group to group and year to year, educational content changes, the priorities of the research discipline shift and the availability of teaching resources and technologies changes.

The upper triangle in figure 3.3 is a depiction of the activities and relationships that inform instructional design and implementation. The center of the diagram describes how students and staff, instructional design and other factors combine in an educational experience and the resulting learning outcomes. The bottom part of the figure visualises the assessment processes designed to provide input to the next design and implementation cycle.

Scaffolding learning becomes increasingly complex as the level of education becomes higher. At the tertiary level learning outcomes, skills and competencies need to be defined and methods for acquiring them debated in the discipline. Debate occurs at several levels. International discussion can result in agreed curriculum standards, such as those defined by the ACM and IEEE for computer science, software engineering and related areas. In other areas curricula and learning outcomes for degree programmes are specified by legislation, or by accreditation bodies such as ABET, and EUR-ACE. These curricula provide high level guidance that is usually supplemented by local decisions within the department.

Curricula define a body of knowlege and the expected outcomes of the education, not the methods by which those outcomes are to be achieved. Realising learning outcomes becomes a much less intimidating process when prior research is available upon which to base instruction. We advocate following a cycle of identifying outcomes, reflecting on how these relate to the discipline and disciplinary knowledge, identifying relevant disciplinary education research, and revisiting the desired outcomes. Following this process leads to instructional designs that are explicit and based on evidence rather than personal opinion.

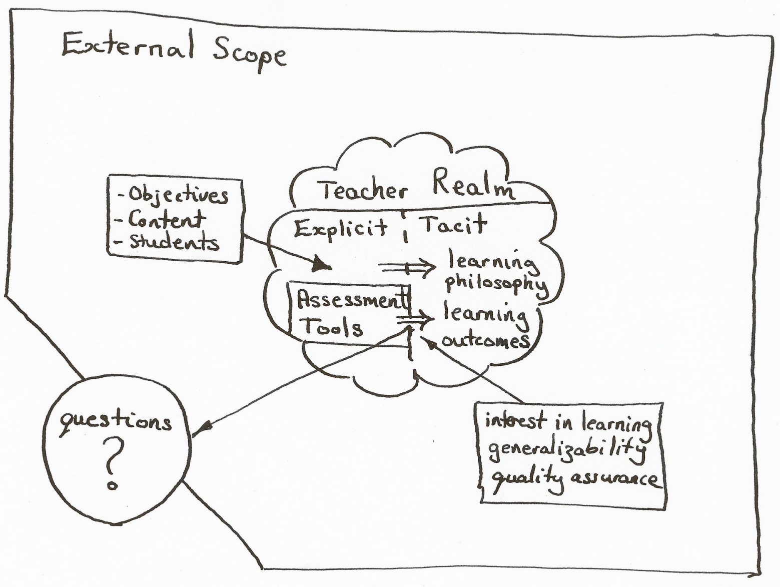

Figure 3.2: Emergence of Uncertainty in Scholarly Teaching Practice

3.2 Understanding learning

Following the informed design process outlined in the previous section makes much more about the conduct of tertiary STEM education explicit. While this offers many advantages contextualising disciplinary education research results also requires thought and reflection. Issues of culture, administrative structure and regulation within the university have an impact that cannot be ignored.

As a result it is not uncomon for teaching and learning issues to arise, even in courses that are based on research that has previously been highly successful. In some cases the issues that arise can be addressed directly using data that has already been collected. However, if questions emerge which the data collected not capture it is natural for teachers to take a step towards disciplinary education research to obtain answers.

The approach that we take to evaluating the impact of innovation in the classroom depends to a large extend on the stakeholder group we wish to convince. As disciplinary experts and higher education professionals, classroom observation of learners may provide sufficient evidence that new approaches produce the desired effects. On the other hand, if one wishes to convince sceptical colleagues, or external accreditation committees of the efficacy and validity of new approaches to instruction more rigorous methods might be required.

The model for scholarly instructional design presented in the previous section may provide useful background material, but in itself may not convince an external observer. Anecdotal evidence and observations may also not hold much weight. The approach taken to collection and analysis of data that support new methods depend on the type of questions that need to be answered. What aspects of learning do we need to collect evidence about?

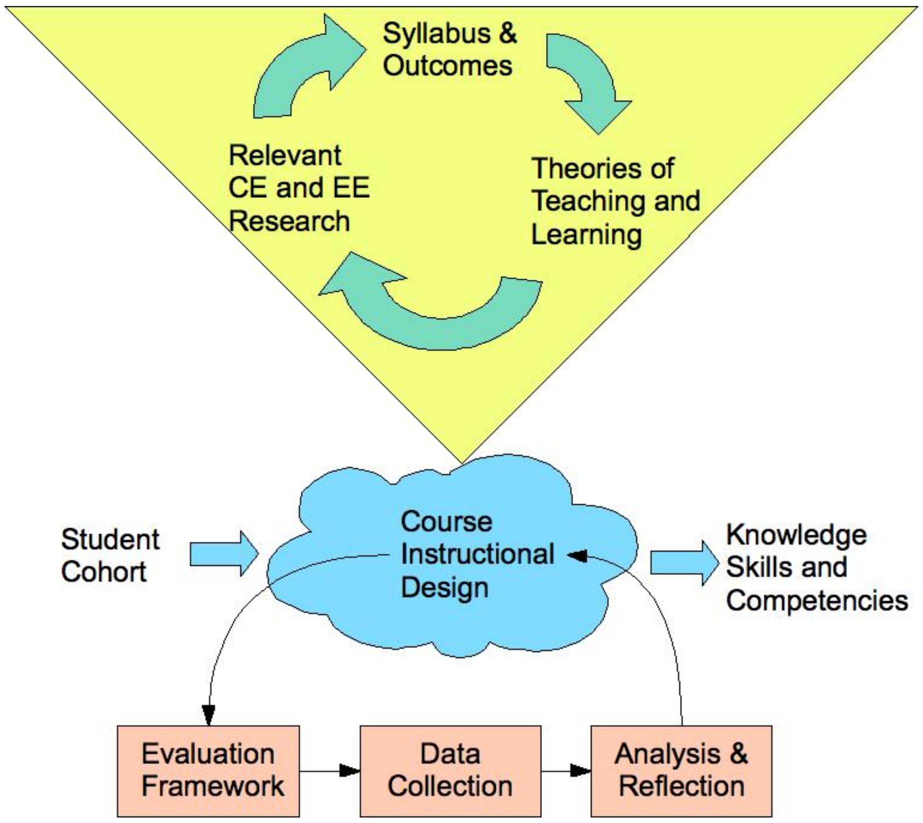
Syllabus & Outcomes
Theories of Teaching and Learning
Relevant CE and EE Research
Student Cohort
Course Instructional Design
Knowledge Skills and Competencies
Evaluation Framework
Data Collection
Analysis & Reflection

Figure 3.3: Developing and assessing instructional design

Part II

Case Study

Chapter 4

An Evidence-based Approach to Teaching Programming

4.1 Introduction

What criteria are used to determine whether or not a computer program or other software product is of "high quality" and meets the "requirements" defined for it? The software engineering community has engaged with questions of quality since the development of early computer systems. There has been considerable focus on elicitation of requirements, rigorous specification approaches, ensuring congruence between the specification and implementation, and demonstrating that the final product fulfills all the requirements of the specification.

Peter Denning addressed this topic in an editorial in Communications of the ACM in 1992 [24], where he criticises the tight technical focus in the criteria of prevalent software quality approaches of the time. He advocated an approach based on customer satisfaction and defined three levels: all basic promises were met; no negative consequences were produced; and the customer is delighted.

Based on wide ranging research and industry experience processes and methodologies for developing and testing software have been specified and evaluated, and now form the basis of project development approaches and tools. In the 1980's and '90's design and specification activity was often highly prioritised, and software engineering courses focused on the waterfall software development approach at many universities. More recently extensions, refinements and alternative approaches have emerged. Examples include the Rational Method, UML, and Agile Development, to name but a few.

But, what priorities emerge at University in situations where students are taught software development skills? How do students go about learning to write reliable and dependable software systems? What challenges to they encounter?

What do they perceive as defining program correctness? It could be argued that despite Denning's admonitions, much of the university education in software development is technical. The outcomes of this focus are evident in the way in which we teach and assess software development activities.

In this paper we examine this issue in the context of early development of conceptions of program and software quality. We use the terms "software" and "program" interchangeably in the paper, following the belief that programs are small, simple software systems. We believe that these early conceptions are highly influential in the formation of individual beliefs about software quality and that these tacit values persist beyond the tertiary education context into professional life.

4.2 Related Research

Incorporating notions of quality into undergraduate education in the computing disciplines is a complex undertaking. Software quality assessment is a sophisticated and complex activity. Defining and assessing software quality is a research area in itself, and addresses both technical and non-technical criteria. Many definitions of quality have been proposed. However, a complete assessment of the state of research in software quality, and how software development principles is well beyond the scope of a single conference paper. This paper examines, in the light of computer science education research, the manner in which early programming courses set the scene for later development of conceptions of quality. Advanced conceptions of software quality go far beyond mere program construction, however, for many software developers their first experiences of quality criteria in relation to software development are obtained in the introductory programming course early in their degree.

Learning to program means different things to different segments of the computing community, so it can be hard to agree on what aspects of quality are paramount. An analysis of the scholarly literature on teaching and learning of programming identifies several schools of thought. Programming, and simple software development can be seen as problem solving, about learning the syntax of the language, about describing data transformations, to name just a few. For a more in depth discussion of some aspects of this issue see Pears et al. [14].

The three bodies of research literature appear to be relevant to developing recommendations for developing learner awareness of, and proficiency with, software quality concepts and practices for bachelors level students. We review each of these and then discuss the implications for university computing education.

4.2.1 Difficulties with learning programming

A significant body of research supports the view that it is hard to learn to program. A good overview of the literature and issues can be found in Robins et al. [25]. After taking programming courses many students have very rudimentary programming, software design, testing and debugging skills [26, 27].

Eckerdal has investigated novice students' conceptions and misconceptions of the notion of object and class [28]. She identifies three types of understanding of each of these concepts based on interview data collected from undergraduate students. The rudimentary conception "class" and "object" is as code structures, the intermediate conceptions distinguish between the role of the object as a program entity and the class as a description of the properties and functionality of objects. The most sophisticated category of understanding is related to modelling physical and conceptual entities.

Another recent US study of programming misconceptions [29] concludes that students do not have a good understanding of the relationship between memory allocation and the notional machine and programming language constructs. They also conclude that most students have no conception of what an object is.

Incorrect understandings of the concept of an object variable in object oriented programming has been studied by Sorva [30] He identifies four categories of understanding, two of which (NamedValue and PlaceForValue) are consistent with the definitions accepted by the object oriented programming community. The other two categories of understanding (MathVariable and PlaceForRef) are "overextensions" or overgeneralisations of the other two. The MathVariable understanding is probably arrived at by attempting to integrate a computing specific concept with a, syntactically very similar, one in mathematics.

4.2.2 Developing holistic system development skills

Learning to program clearly involves more than mastering the language syntax and semantics, it appears to also require ability to relate goals to coding activity and move between a holistic and low level view of software functionality during software development. Studies dealing of the fragile nature of novice understanding of programming language syntax and semantics have been conducted for both traditional imperative programming and object oriented programming paradigms.

Novice inability to attack programming problems holistically and systematically has been investigated quite early by Soloway [31], who theorised that this was related to inability to articulate goals and devise coding plans to achieve these goals.

De Raadt et al. [32], building upon the idea of goal/plan analysis, argued that code development strategies need to be more explicitly taught. They assessed student's ability to apply "plans" when solving a previously published and analysed averaging problem [31]. Their work culminated in a new approach to teaching, in which strategy is an explicit part of the curriculum [33, 34].

Bennedsen and Caspersen argue that an important part of learning to program in a structured manner is understanding the expert processes involved. They note that textbooks do not address the need to make tacit expert process explicit for learners and propose process recordings (narrated recordings of programming sessions) to fill this gap. They identify seven areas where such process information has significant value to learners. Five of these areas are related to developing quality software, incremental development, testing, refactoring, debugging and model development. Two deal with programming support, learning the IDE and using online documentation.

Defining what ”programming thinking” involves is discussed by Eckerdal et al. They concluded that students need to develop a conception of software development and programming as “... a way of thinking, which enables problem solving, and which is experienced as a ‘method’ of thinking” [35, p.7]

T.R.G. Green [36] discusses the cognitive complexity of programming languages and how this can predispose less experienced programmers to focus on syntactic details instead of the broader functional goals of the software development activity. He notes that much of the activity in program development is about incremental improvement, and that most integrated development environments (IDE’s) provide good support for this development approach. In contrast to structured software development, learners tend to develop their software as an integral aspect of exploring the functionality of the programming language and compilation and debugging tools. This type of programming approach is not well supported by most commercial IDE’s, which raises interesting questions about what tools might be more appropriate.

4.2.3 Student conceptions of correctness and quality

Definitions and perceptions of the nature of program correctness and quality among students have been the subject of several recent studies.

Kolikant and Mussai’s [37, 38] 2008 investigation of student conceptions of software/code correctness [38] is based on data collected from 159 students who were asked to analyse error-free and erroneous algorithms. Followup interviews were also conducted with seven students. They concluded that students viewed programs as collections of lines of code and programming operations, each of which could be correct or partially correct.

Stamouli and Huggard [39] conducted a year long study in which they followed sixteen students developing understanding of programming and program correctness. Each student was interviewed on four occasions throughout the year and the interviews subjected to a Phenomenographic analysis. Their categories of understanding are very similar to those of Eckerdal et al. and demonstrate a strong student preoccupation with language syntax.

4.3 Discussion

Quality exists at many levels, readable and structured code, logical decomposition, emphasis on interfaces and interface specification and use of appropriate processes and tools to name but a few. We are focusing on incorporating some of the lower level quality attributes into education for novice and intermediate programmers.

Understanding how learners go about learning to program and what misconceptions and difficulties they encounter provides insight into what they perceive and what they experience when they develop software.

4.3.1 Can't see the wood for the trees

We conclude from the literature reviewed in the previous section that many of the difficulties students encounter, and the (mis)conceptions they develop, indicate that early learning in programming is dominated by syntactic concerns and that most students lack an holistic view of program function. A student quote from Stamouli and Huggard serves to highlight this.

> "Liam3: It's about getting a couple of books, really... Well it is the language that you are interested in anyway and the syntax of the language is what you learn from the book. Then you have to mess around with the syntax and the features of the language and having fun you learn more, really [. . .]. So what you can do depends really on the language and this is, really, what you learn from a book, the syntax of the language, that's all you need anyway." [39, p.111]

Bennedsen and Caspersen report that ".. students do not know when it is advantageous to re-factor a program; they consider the job done when the program can compile and run." [40, p.3]. Studies of misconceptions and problems learning to program have also identified a strong student focus on syntax in their understanding of object and class [28, 39]. While Sorva [30] demonstrates that there is significant confusion about fundamental concepts such as the nature of variables, perhaps in part due to an overlap in terminology with mathematics. All of these factors, combined with the fact that current textbooks also focus on language features rather than process, contribute to an environment where students find to difficult to learn the skills and processes needed to write good quality software.

Readable clearly structured code is widely accepted as being easier to debug and extend, but there is much anecdotal evidence to suggest that students do not share this perception. Rather than dividing programs into logical units, an approach which has the added advantage of reducing the cognitive load of the task, students have a demonstrated tendency to focus on code at the operation or line level, showing little understanding of the program as a whole.

> "Novices are more likely to give line by line explanations of code than to describe the overall purpose of a piece of code, suggesting that they do not possess the strategies used in the generation of the code." [38, p.1]

This view is also supported by the research of McCracken et al. and Lister et al. and many others who have continued this work [26, 27]. The conclusion to be drawn from this body of research is that many students at the conclusion of an introductory programming course when given a functional description are unable to write a piece of software that meets the requirements. Indeed many students are not able to explain what a piece of code does in a more advanced manner than the line by line approach described above. An interesting discussion of how students react to being asked to "explain" code in English can also be found in Simon [41]

One could propose, drawing on the work of Soloway, de Raadt et al., Bennedsen and Caspersen, and Hilburn and Towhidnejad (and probably others the author is not aware of), that we should refocus the curriculum to emphasise process and quality issues from the start. This is certainly something to be considered, as the new demands of the software industry and increasing availability of code fragments and examples through the internet appear (at least anecdotally) to be changing the way in which students develop code.

The author has personal experience of students cutting and pasting in sections of code from the web and not realising that they need to change the variable names to make them consistent with the rest of the code. When the issue was discussed the students explained that they had assumed the code they pasted in was ”generic” and would not require any changes in order to function.

4.3.2 Student conceptions of correctness

Another conclusion we can draw from the literature is that program behaviour is poorly understood and taught, and that notional machine concepts seem to be important in this regard [29].

Research surrounding the visualisation tool Jeliot [42, 43] deals with using a detailed notional machine visualisation in teaching and programming activities. Some of their results imply that good conceptualisations of memory allocation and stack operations assist students in developing a more sophisticated understanding of program operation.

There are also indications that alignment of software quality attributes with assessment practices in early phases of programming and software development education at university may be poor.

A quote from Kolikant and Mussar serves to reinforce the conclusion that students often perceive programs at the syntactic (line by line command) level.

> “We found that students conceptualized program correctness as the sum of the correctness of its constituent operations and, therefore, they rarely considered programs as incorrect. Instead, as long as they had any operations written correctly students considered the program ‘partially correct’.” [38, p.1]

Their conclusion is important for the teaching of software/program quality at all levels, but especially for novice programmers. It appears that misalignment of assessment with programming behaviours and practices that enhance software quality create a destructive programming culture early in degree programmes.

> “We suggest that this conception is a faulty extension of the concept of a program’s grade, which is usually calculated as the sum of points awarded for separate aspects of a program. Thus school [University] (unintentionally) nurtures students’ misconception of correctness. This misconception is aligned with students’ tendency to employ a line by line verification method – examining whether each operation is translated as a sub-requirement of the algorithm – which

> is inconsistent with the method of testing that they formally studied." [38, p.1]

Other definitions of correctness can also be identified in the literature. A common theme, connected to student preoccupation with syntax, is that a program is correct when it compiles. Evidence of this view includes the earlier quote in this section from Bennedsen and Caspersen. This view is clearly more widespread than a single study however. Stamouli and

> "The findings also show that more than half of the population of the study does not develop a complete and mature understanding of learning to program such as the ones described in categories 5 and 6. A similar relationship holds for the understanding of program correctness. The interviews were held almost at the end of the academic year, so there were some students who completed the programming course and who still believed that learning to program in the object oriented paradigm is purely about learning the syntax of the language and that a program is correct when it is free from syntactical errors." [39, p.114]

This should prompt us to revisit some of our approaches to assessment. A precondition for being assessed in many programming assignments in courses at university is that the program compiles and runs. Instructors then start to test the code for functional compliance with the specification. However, this sends a clear message that educators place high value on the fact that the code compiles and runs. Is this really what we want to convey?

Applying principles from constructive alignment [4] suggests that more emphasis in grading should probably be placed on quality attributes and students demonstrating code development based on desirable work practices. This harmonises with the approach suggested by Hilburn and Towhidnejad where quality processes are an explicit element of the curicculum and assessment.

4.3.3 Making processes explicit

Testing is accorded little importance by students despite the professional emphasis on this, is there misalignment here in terms of how we assess student work?

Hilburn and Townhidnejad [44] discussed how to improve the visibility of quality issues in the undergraduate curriculum. They propose a quality model, and show how it can be applied in undergraduate education. Aspects of this model may well be of value in other contexts where the teaching of quality as an integral part of the course is desirable.

Patton and McGill [45] advocate longitudinal portfolios of programming work and artifacts in combination with the use of automated software quality metrics as a way to evaluate student progress in developing quality software throughout their university career. This approach also allows both student and educators to assess development of professional skills and appreciation of the role of quality and process in developing software systems.

4.4 Recommendations

Drawing on the discussion in the previous section we propose three fundamental recommendations for improving conceptions of software quality among undergraduate students. It is important to emphasise that these recommendations emerge from our analysis of the existing computing education research literature, rather than arising from personal opinions about how programming or software development should be taught.

Based on our synthesis of prior research we recommend that in order to enhance the opportunity for students to learn how to develop quality software at university educators should.

- Adopt an explicit holistic goal/plan based approach to code development that commences in the introductory programming course and continues throughout the software development courses in the degree programme.
- Design in introduce formative assessment strategies which improved the alignment of key aspects of quality with high grade achievement, and break the current assessment trends that can be demonstrated to establish beliefs about "partial correctness" of code.
- Include testing and debugging as explicit parts of the curriculum.

4.5 Applying Theory to Practice

Building on this theory a course in introductory object oriented programming was designed and run at Uppsala University, Sweden. The design is presented and related to the earlier discussion on conceptions of quality, as a concrete example of how our research and analysis can be used in evidence based curriculum development.

4.5.1 Instructional Design

The curriculum and instruction approach described here follows the explicit constructivist approach prevalent in much of computer science education [46]. We hypothesise that experiences related to the practice of programming helps to build familiarity with language structures and concepts. The importance of this for the construction of programming knowledge in computing has been previously emphasised by Eckerdal [47]

There is a huge debate in computing about how to teach object oriented programming skills, and when to do this in a university programming course sequence [48–50]. Our conclusion from a detailed examination of the literature is that there is no approach which research demonstrates to be more effective [14], so a procedural and control structures approach was taken.

The material was presented in the following order.

1. Sequence

2. Alternation
3. Repetition
4. Modularity
5. Parameterization of modules, functions and procedures.

We then relate these understandings to the syntax of other imperative languages. This is done by interspersing lecture and discussion presentations with hands-on implementation and code exploration exercises. A ten week introductory phase consisted of one 90 minute full class interactive programming demonstration by the instructor, followed by a 90 minute coding laboratory and additional homework exercises.

A further ten weeks were devoted to group projects. Project proposals were prepared by the programming groups based on instructions given in the lecture. Each group had a week to prepare the proposal, and then presented it to the rest of the class and the lecturer in a 10 minute presentation. If the project was sufficiently complex, involved at least two classes, and stored and retrieved complex information in a primitive database [1], it was approved, and the group began implementation. During the ten project weeks the 90 minute laboratory sessions were used by the lecturer to interact with the groups and observe their progress. Groups were required to attend and demonstrate their code regularly to enable the lecturer to take notes on the learning progress of individuals. These notes, together with a final group and individual interview of 30 minutes and 10 minutes respectively provided the data used to determine the final grade of each student.

4.5.2 Learning Approach

Dialogue and continuous explicit reinforcement of tacit quality attributes are the main focus of the course. However, student motivation plays a vital role. Students must perceive the value and rewards associated with practices that lead to quality outcomes in a direct and tangible manner if we follow a constructive alignment approach [4].

Students are asked right at the beginning of the course about what they want to achieve and we relate this to the content and structure of the programming to be undertaken. Malmi in Bennedsen, Caspersen and Kölling [51] has investigated student perceptions of programming assignments. While offering recommendations for areas that are motivating, he also notes that allowing students to define their own projects, within the constraints of what is required in order to achieve appropriate learning outcomes, produces higher overall motivation.

Definition of individual programming goals (through a reflective exercise) helps individuals to enunciate personal goals, and reflect on their learning. Reflection and engagement are strongly linked to predisposing students to adopt

[1]The database was to be implemented as a separate class, and encapsulate a simple data structure such as an array or linked list.

deep learning tendencies [8, 52]. We also adopt practices based on the research of Barker et al. [53] designed to promote an open and constructive class communication culture, which has also been linked to improved learning outcomes for CS students. Specific topics that are regularly revisited in all programming contexts are, modularity, code readability, the need for redesign and refactoring. We also discuss coupling and cohesion in informal terms.

We hypothesised that if we could encourage students to articulate the reasoning behind their programming activities that this would help students to gain a more holistic view of software construction. Peer programming, governed by strict instructions to reason aloud, and reminders to swap driver/navigator roles regularly, was used to strengthen self perception of competence. and has also been demonstrated to encourage peer learning and promote improved outcomes in programming student cohorts [54–56].

Lectures were designed to address the student learning needs and questions that arose in the hands on programming sessions. Lecture presentation was designed to help students to discern key facets of the "expert tacit knowledge" associated with programming and software development. Opportunities to observe the manner in which code is written and debugged by experts are few in most programming courses. Like experts in many other areas the techniques expert programmers use to develop code and correct errors are often tacit [57]. The relevant techniques and skills are certainly not visible in the normal lecture environment where pre-tested code is presented in a sequential manner, or in whole blocks on an overhead slide. Providing learners with an opportunity to observe expert practice is an important part of assisting them in quickly acquiring a sophisticated approach to code development and debugging.

Students were also provided with a large body of code examples, both our own and via online tutorials and repositories. This rich resource pool provides good "exemplars", creating a context for discerning what constitutes "good code" and providing resources and support for exploring program behaviour in a visualisation environment. To support student's developing an intuitive understanding of the von Neumann machine model that underlies much of sequential computer programming[2] we used Jeliot [43].

4.5.3 Constructive Alignment of Assessment

The alignment of assessment should reward students for demonstrating an appreciation of aspects of software quality. We have structured the assessment to directly reward behaviour and code development practices that are widely accepted to improve software quality. The grades we give are designed to reflect the value we place on understanding that quality is a valued and appreciated part of the course activities. As a result of this philosophy we formulated the course learning outcomes as follows.

After having passed the course a student will have demonstrated that they are consistently capable of doing the following.

[2]As distinct from parallel and multi-core programming, which requires significantly different types of machine model and abstractions in order to reason about program behaviour

Goal	Practical sessions	Assignment 1	Assignment 2	Programming project
1		Analysis	Analysis	Design and req. spec.
2				Apply OO or decomposition
3		arrays	lists	more complex data structures, Java lib.
4	exposure	intro to testing	develop testing	independent testing
5	exposure/practice	simple control	functions and param	use of OO and imperative constructs
6	exposure	guided develop.	independent develop.	group develop.
7	discuss	-	-	apps in other domains
8	exposure	-	may choose alt. lang	-
9	exposure	Java lib.	Java lib.	Open source code use encouraged
10	discussion	pair coding	pair coding	group coding (4 students)
11	homework exercises			

Table 4.1: Association of Learning outcome with Assessment Instrument

1. Analyse fairly simple problems, formulate an appropriate algorithmic solution, and express it as a simple program specification.

2. Formulate and articulate strategies for solving larger programming tasks.

3. Write functioning programs that use basic data-structures, for instance arrays, different sorts of lists and binary trees.

4. Collect information on, and reason about, the cause of syntactic, logical and runtime errors in a systematic manner.

5. Explain the execution and purpose of programs consisting of common imperative programming constructs (sequences, choices, loops, functions) in an object oriented language, such as Java.

6. Demonstrate and understanding of the basics of good program construction and familiarity with the object oriented programming style.

7. Discuss and give examples of how programming is used to solve problems in other disciplines.

8. Demonstrate ability to apply their programming knowledge to programming in another closely related imperative language (for instance, Python, C or C++).

9. Re-use code developed by others in their own programming.

10. Explain the different roles/activities associated with developing programs.

11. Explain in high level terms the structure and operation of a von Neumann style computer.

Measuring some of these outcomes requires a continuous assessment approach that is integrated into the day to day teaching activities. To achieve this we replaced the assignment and examination assessment model with a continuous feedback and assessment scheme based on a programming portfolio evaluated by observing students activity in three major course areas, the weekly 90 minute sessions, in person demonstration and explanation of homework programming exercises, and the design, implementation and verbal explanation of the larger software system developed by groups of 4-6 students. The linking between the course goals and the assessment instruments is summarised in Table 4.1.

The outcomes of this approach were encouraging. Students who completed the course successfully produced well documented and structured code, far above the quality often produced by a first year student. The teams worked successfully in teams, decomposed the system functionality and were able to program independently and integrate object oriented components of the final system to produce a working suite of software at the conclusion of the 10 week project. Over 80 percent of students who took the course were considered to have met the specified learning goals and received a pass mark or better.

Many students in their final reflective exercises mentioned that they valued the project aspects of the approach.

> ”The examination as a project was a successful idea, I learned a lot more during those weeks than studying to a written examination.”

However, there were also students who were quite critical of the approach and would have preferred a more traditional structure. The following quote (translated into English from Swedish) illustrates one concern.

> ”Det kändes som att betygssättningen var lite godtycklig då vi inte hade haft [läraren] på våra lektioner, och svårighetsgraden på projektet verkade bestämma det mesta. Det är det svårt att kompromissa om det i en grupp då alla ligger på olika nivå och ambitionsnivå.”

> ”It felt like the grading was a little arbitrary given that we had not had [the lecturer] in our laboratory session much, and where it seemed that the level of difficulty of the final project code was the major factor influencing the final grade. This is difficult when group members have different ability levels and aspirations in terms of final grade.”

In fact the grades were arrived at after a detailed interview with both the project groups and each individual who took the course. This interview asked students to identify and reason about the function of code that they had written for the system, and also to reason aloud about design tradeoffs and concepts that are important for object oriented programming. Final grades were awarded based on the outcomes of these interviews and an inspection of the project source code.

4.6 Conclusion

Better approaches to educating software engineering professionals are needed. To achieve this we need to consider how to align professional and community values more clearly with our curriculum.

The aim of this paper has been to discuss what is needed to enhance appreciation of quality, and how is that to be conveyed to young aspiring professionals during their education? The result of our survey and analysis of relevant research is three major recommendations for curriculum changes. We encourage educators to consider these recommendations seriously when designing future courses in programming and software development at university.

Taking these recommendations into account we presented an overview of a teaching approach developed for an introductory course in computer programming at Uppsala University. This example shows more clearly how we believe our principles can be used in practice, and how curriculum and instructional design can be informed by teaching and learning research in computer science.

More research is clearly needed to identify a greater range of alternative assessment approaches that align grading outcomes with engagement in practices that promote the development of quality software. One avenue to investigate, is how grading schemes can capture a more holistic view of the system being graded. It also seems important to develop grading/assessment schemes that reward learners for using quality assurance processes and rigorous testing during software development.

Chapter 5

Using Action Research to Evolve a Service Learning Course

Bibliography

[1] L. I. Brown, “Diversity: The challenge for higher education,” *Race Ethnicity and Education*, vol. 7, no. 1, pp. 21–34, 03 2004. [Online]. Available: http://www.eric.ed.gov/ERICWebPortal/detail?accno=EJ681255

[2] J. Biggs, *Teaching for Quality Learning at University (2nd edition).* Society for Research into Higher Education, Open University Press, 2003.

[3] E. Schofer and J. W. Meyer, “The worldwide expansion of higher education in the twentieth century,” *American Sociological Review*, vol. 70, no. 6, pp. 898–920, 12 2005. [Online]. Available: http://www.jstor.org/stable/4145399

[4] J. Biggs, “Enhancing teaching through constructive alignment,” *Higher Education*, vol. 32, no. 3, pp. 347–364, 1996.

[5] E. Boyer, *Scholarship Reconsidered: Priorities of the Professoriate.* Hillsdale, NJ: Carnegie Foundation for the Advancement of Teaching and Josey-Bass, 1990, 1997.

[6] C. E. Glassick, M. T. Huber, G. I. Maeroff, and E. L. Boyer, *Scholarship assessed: evaluation of the professoriate.* San Francisco: Jossey-Bass, 1997.

[7] M. Prosser and K. Trigwell, *Understanding Learning and Teaching: The experience in higher education.* Buckingham: Open University Press, 1999.

[8] G. Gibbs and M. Coffey, “The Impact Of Training Of University Teachers on their Teaching Skills, their Approach to Teaching and the Approach to Learning of their Students,” *Active Learning in Higher Education*, vol. 5, no. 1, pp. 87–100, 2004. [Online]. Available: http://alh.sagepub.com/cgi/content/abstract/5/1/87

[9] K. Trigwell and S. Shale, “Student learning and the scholarship of university teaching,” *Studies in Higher Education*, vol. 29, no. 4, pp. 523–536, 2004. [Online]. Available: http://www.ingentaconnect.com/content/routledg/cshe/2004/00000029/00000004/art00007

[10] F. Marton and R. Säljö, "On qualitative differences in learning," *British Journal of Educational Psychology*, vol. 46, pp. 4–11, 115–127, 1976.

[11] M. Prosser and K. Trigwell, "Confirmatory factor analysis of the approaches to teaching inventory," *British Journal of Educational Psychology*, vol. 76, pp. 405–419, 2006.

[12] E. Martin, M. Prosser, K. Trigwell, P. Ramsden, and J. Benjamin, "What university teachers teach and how they teach it." *Intructional Science. Special issue: Teacher Thinking, Beliefs and Knowledge in Higher Education*, vol. 28, no. (5-6), pp. 387–412, 2000.

[13] D. Kember and K.-P. Kwan, "Lecturers' approaches to teaching and their relationship to conceptions of good teaching," *Instructional Science*, vol. 28, pp. 469–490, 2000, 10.1023/A:1026569608656. [Online]. Available: http://dx.doi.org/10.1023/A:1026569608656

[14] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming," *SIGCSE Bull.*, vol. 39, no. 4, pp. 204–223, 2007.

[15] M. M. Waldrop, "Why we are teaching science wrong, and how to make it right." *Nature*, vol. 523, no. 7560, p. 272, 2015.

[16] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, "Active learning increases student performance in science, engineering, and mathematics," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8410–8415, 2014. [Online]. Available: http://www.pnas.org/content/111/23/8410.abstract

[17] M. Guzdial. (2015, August) Be it resolved: Teaching statements must embrace active learning and eschew lecture. [Online]. Available: http://cacm.acm.org/blogs/blog-cacm/190767-be-it-resolved-teaching-statements-must-embrace-active-learning-and-eschew-lecture/fulltext

[18] S. Scott. (2017, March) Should we lose the lecture? [Online]. Available: https://medium.com/stanford-magazine/should-we-lose-the-lecture-76a186797573

[19] K. Trigwell, M. Prosser, and F. Waterhouse, "Relations between teachers' approaches to teaching and students' approaches to learning," *Higher Education*, vol. 37, pp. 57–70, 1999.

[20] N. Entwistle, "Conceptions of Learning and the Experience of Understanding: Thresholds, Contextual Influences, and Knowledge Objects," in *Reframing the conceptual change approach in learning and instruction*, S. Vosniadou, A. Baltas, and X. Vamvakoussi, Eds. Amsterdam, The Netherlands: Elsevier, 2007, ch. 11.

[21] D. Kember, "Beliefs about knowledge and the process of teaching and learning as a factor in adjusting to study in higher education," *Studies in Higher Education*, vol. 26, no. 2, pp. 205–221, February 2001. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/03075070120052116

[22] K. Trigwell, E. Martin, J. Benjamin, and M. Prosser, "Scholarship of teaching: A model," *Higher Education Research & Development*, vol. 19, no. 2, pp. 155 – 168, 2000.

[23] L. S. Shulman, "Taking learning seriously," *Change: The Magazine of Higher Learning*, vol. 31, no. 4, pp. 10–17, 1999.

[24] P. J. Denning, "Editorial: what is software quality?" *Commun. ACM*, vol. 35, no. 1, pp. 13–15, 1992.

[25] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: a review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.

[26] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs students," *SIGCSE Bull.*, vol. 33, no. 4, pp. 125–180, 2001.

[27] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A multi-national study of reading and tracing skills in novice programmers," in *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education.* New York, NY, USA: ACM, 2004, pp. 119–150.

[28] A. Eckerdal and M. Thuné, "Novice java programmers' conceptions of "object" and "class", and variation theory." in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education,*, J. J. Dougherty, Ed. Lisbon, Portugal. New York: ACM Press, 2005, pp. 89 – 93.

[29] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proceedings of the 41st ACM technical symposium on Computer science education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, pp. 107–111. [Online]. Available: http://doi.acm.org/10.1145/1734263.1734299

[30] J. Sorva, "Investigating incorrect understandings of a CS concept," in *Second Nordic Workshop on Phenomenography in Computing Education Research.* Uppsala University, 2008.

[31] E. Soloway, "Learning to program = learning to construct mechanisms and explanations," *Communications of the ACM*, vol. 29, no. 9, pp. 850–858, 1986.

[32] M. de Raadt, M. Toleman, and R. Watson, "Training strategic problem solvers," *SIGCSE Bull.*, vol. 36, no. 2, pp. 48–51, 2004.

[33] M. de Raadt, "A review of australasian investigations into problem solving and the novice programmer," *Computer Science Education*, vol. 17, no. 3, pp. 201–213, 2007. [Online]. Available: http://www.informaworld.com/10.1080/08993400701538104

[34] ——, *Teaching Programming Strategies Explicitly to Novice Programmers: Can the way we teach strategies improve novice outcomes?* Saarbrücken, Germany, Germany: VDM Verlag, 2009.

[35] A. Eckerdal and A. Berglund, "What does it take to learn 'programming thinking'?" in *Proceedings of The First International Computing Education Research Workshop.* ACM Press, 2005, pp. 135–143.

[36] T. R. G. Green, "Instructions and descriptions: some cognitive aspects of programming and similar activities," in *AVI '00: Proceedings of the working conference on advanced visual interfaces.* New York, NY, USA: ACM Press, 2000, pp. 21–28.

[37] Y. B.-D. Kolikant, "Students' alternative standards for correctness," in *The Proceedings of the First International Computing Education Research Workshop.* ACM Press New York, NY, USA, 2005, pp. 37–43.

[38] Y. B.-D. Kolikant and M. Mussai, ""So my program doesn't run!"definition, origins, and practical expressions of students' (mis)conceptions of correctness," *Computer Science Education*, vol. 18, no. 2, pp. 135–151, 2008. [Online]. Available: http://www.informaworld.com/10.1080/08993400802156400

[39] I. Stamouli and M. Huggard, "Object oriented programming and program correctness: The students' perspective," in *Proceedings of the Second International Computing Education Research Workshop (ICER)*, R. Anderson, S. Fincher, and M. Guzdial, Eds. Canterbury, U.K. New York: ACM Press, 2006, pp. 109 – 118.

[40] J. Bennedsen and M. E. Caspersen, "Revealing the programming process," in *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education.* New York, NY, USA: ACM, 2005, pp. 186–190.

[41] Simon, "A note on code-explaining examination questions," in *Koli '09: Proceedings of the 9th International Conference on Computing Education Research*, A. Pears and C. Schulte, Eds., 2009.

[42] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, “Visualizing programs with Jeliot 3,” in *AVI ’04: Proceedings of the working conference on Advanced visual interfaces.* New York, NY, USA: ACM Press, 2004, pp. 373–376.

[43] N. Ragonis and M. Ben-Ari, “On understanding the statics and dynamics of object-oriented programs,” in *SIGCSE ’05: Proceedings of the 36th SIGCSE technical symposium on Computer science education.* New York, NY, USA: ACM Press, 2005, pp. 226–230.

[44] T. B. Hilburn and M. Townhidnejad, “Software quality: a curriculum postscript?” *SIGCSE Bull.*, vol. 32, no. 1, pp. 167–171, 2000.

[45] A. L. Patton and M. McGill, “Student portfolios and software quality metrics in computer science education,” *J. Comput. Small Coll.*, vol. 21, no. 4, pp. 42–48, 2006.

[46] M. Ben-Ari, “Constructivism in computer science education,” *Journal of Computers in Mathematics and Science Teaching*, vol. 20, no. 1, pp. 45–73, 2001.

[47] A. Eckerdal, “Novice programming students’ learning of concepts and practise,” Ph.D. dissertation, Uppsala UniversityUppsala University, Division of Scientific Computing, Numerical Analysis, 2009.

[48] R. E. Pattis, “The “procedures early” approach in cs 1: a heresy,” in *SIGCSE ’93: Proceedings of the 24th SIGCSE technical symposium on Computer science education.* New York, NY, USA: ACM Press, 1993, pp. 122–126.

[49] E. Howe, M. Thornton, and B. W. Weide, “Components-first approaches to CS1/CS2: principles and practice,” in *SIGCSE ’04: Proceedings of the 35th SIGCSE technical symposium on Computer science education.* New York, NY, USA: ACM Press, 2004, pp. 291–295.

[50] R. Lister, A. Berglund, T. Clear, J. Bergin, K. Garvin-Doxas, B. Hanks, L. Hitchner, A. Luxton-Reilly, K. Sanders, C. Schulte, and J. L. Whalley, “Research perspectives on the objects-early debate,” in *ITiCSE-WGR ’06: Working group reports on ITiCSE on Innovation and technology in computer science education.* New York, NY, USA: ACM Press, 2006, pp. 146–165.

[51] J. Bennedsen, M. Caspersen, and M. Kölling, *Reflections on the Teaching of Programming.* Springer-Verlag, 2008, vol. 4821.

[52] S. E. George, “Learning and the reflective journal in computer science,” in *ACSC ’02: Proceedings of the twenty-fifth Australasian conference on Computer science.* Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2002, pp. 77–86.

[53] K. Garvin-Doxas and L. J. Barker, “Communication in computer science classrooms: understanding defensive climates as a means of creating supportive behaviors,” *J. Educ. Resour. Comput.*, vol. 4, no. 1, p. 2, 2004.

[54] L. Williams and R. L. Upchurch, “In support of student pair-programming,” *SIGCSE Bull.*, vol. 33, no. 1, pp. 327–331, 2001.

[55] C. McDowell, B. Hanks, and L. Werner, “Experimenting with pair programming in the classroom,” in *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2003)*, June 30 - July 2 2003. [Online]. Available: http://www2.umassd.edu/SWPI/xp/pairprogramming/private/iticse03final.pdf

[56] N. Jacobson and S. K. Schaefer, “Pair programming in cs1: overcoming objections to its adoption,” *SIGCSE Bull.*, vol. 40, no. 2, pp. 93–96, 2008.

[57] R. Mancy and N. Reid, “Using interviews to investigate implicit knowledge in computer programming,” in *ICLS ’06: Proceedings of the 7th international conference on Learning sciences.* International Society of the Learning Sciences, 2006, pp. 460–466.