A Motivation Guided Holistic Rehabilitation of the First Programming Course

UOLEVI NIKULA, Lappeenranta University of Technology
ORLENA GOTEL, Independent Researcher
JUSSI KASURINEN, Lappeenranta University of Technology

It has been estimated that more than two million students started computing studies in 1999 and 650,000 of them either dropped or failed their first programming course. For the individual student, dropping such a course can distract from the completion of later courses in a computing curriculum and may even result in changing their course of study to a curriculum without programming. In this article, we report on how we set out to rehabilitate a troubled first programming course, one for which the dropout statistic and repercussion was evident. The five-year longitudinal case study described in this article began by systematically tracking the pass rate of a first programming course, its throughput, as proposed by the Theory of Constraints. The analyses of these data indicated three main problems in the course: programming discipline difficulty, course arrangement complexity, and limited student motivation. The motivation problem was approached from the Two-Factor Theory point of view. It investigated those factors that led to dissatisfaction among the students, the hygiene factors, and those factors that led to satisfaction, the intrinsic and extrinsic motivators. The course arrangement complexity was found to be a hygiene factor, while the lack of extrinsic and intrinsic motivators contributed to the high dropout rates. The course improvement efforts made no attempt to change the inherent characteristics of the programming discipline, but introduced holistic changes in the course arrangements over a five-year period, from 2005 to 2009, to eliminate the hygiene factors and to increase motivational aspects of the course. This systems approach to course improvement resulted in an increase in the pass rate, from 44% prior to the changes to 68% thereafter, and the overall course atmosphere turned positive. This paper reports on the detailed changes that were made and the improvements that were achieved over this five-year period.

Categories and Subject Descriptors: K.3.2 [**Computers and Information Science Education**]: *Computer science education; curriculum*

General Terms: Human Factors, Performance

Additional Key Words and Phrases: CS1, course redesign, systems approach, Theory of Constraints, hygiene factors, intrinsic and extrinsic motivators, Two-Factor Theory

ACM Reference Format:
Nikula, U., Gotel, O., and Kasurinen, J. 2011. A motivation guided holistic rehabilitation of the first programming course. ACM Trans. Comput. Educ. 11, 4, Article 24 (November 2011), 38 pages.
DOI = 10.1145/2048931.2048935 http://doi.acm.org/10.1145/2048931.2048935

1. INTRODUCTION

Since the 1970s, introductory programming in a high-level language has been offered as the first course in the overwhelming majority of computer science departments in the U.S. (cf. Furugori and Jalics [1977]). In 1999, it was estimated that more than two

Authors' addresses: U. Nikula and J. Kasurinen, Department of Information Technology, Lappeenranta University of Technology, Finland; email: Uolevi.nikula@lut.fi; O. Gotel, Independent Researcher, New York.

DOI 10.1145/2048931.2048935 http://doi.acm.org/10.1145/2048931.2048935

million students started computing studies, and all of them enrolled in introductory programming courses [Bennedsen and Caspersen 2007]. There is little evidence of this situation having changed much over recent years. However, problems associated with the teaching and learning of introductory programming have also been reported since the 1980s [Sleeman 1986], and this has been a popular topic of ongoing research ever since (e.g., Ebrahimi [1994]; Jenkins [2002]; Kinnunen et al. [2007]; Lahtinen et al. [2005]; Mow [2008]). The implications of these problems were expressed succinctly by Bergin and Reilly [2005]: "It is well known in the Computer Science Education (CSE) community that students have difficulty with programming courses and this can result in high drop-out and failure rates." Bennedsen and Caspersen [2007] estimate that of the plus two million students starting computer science studies in universities and colleges all over the world in 1999, 33%, or 650,000, dropped or failed their first programming course.

In this article, we report on a study surrounding the rehabilitation of a troubled first programming course at Lappeenranta University of Technology (LUT), undertaken between 2005 and 2009, with 157 to 249 students enrolled annually. In 2005, the Fundamentals of Programming course (often referred to as CS1 by many institutions) was offered in two variants: course A for Information Technology (IT) majors and other students who needed to acquire basic programming skills; and course B for those students who needed to know the basics of programming in order to make decisions about its use and resourcing (e.g., management students). We call the course "troubled" since, prior to 2005: (1) the three previous implementations of the course had pass rates of between 44% and 47%; (2) the programming skills of the students were reported as deficient by the instructors of the follow-on courses; and (3) the students appeared dissatisfied with the course. Any one of these issues, if experienced at the onset of a student's university studies, could lead to a delay in graduating, to a curricula change, or to dropping out of college all together, three outcomes that we did not want students to experience. We therefore set out to rehabilitate the first programming course by examining the reasons for the low pass rate and then by planning and managing a holistic course revision, aiming at an 80% pass rate.

The study started by considering the course as a system, with the pass rate as its throughput. This measure was based on the enrolled and graded students, and the objective was to examine and address those bottlenecks that constrained the throughput, as suggested by the Theory of Constraints [Goldratt 1994; Goldratt and Cox 2004]. As typical for improvement initiatives such as the Shewhart cycle [Deming 1990], the constraint identification process needs to be repeated after each constraint has been removed which, in our case, resulted in five replicated cases studies during the period from 2005 to 2009. During this five-year period, the course arrangements were revised in a holistic manner to cover all the key areas identified. The largest changes included a curricula level unification of the two course variants and a revision of the introductory programming course sequence to form a more coherent whole. The course level changes included a move from the C programming language to Python, the development of new study materials, and a redesign of all the course assignments.

Although the study started from the need to rehabilitate a troubled course with a low pass rate, it ended up being a study of student motivation. Numerous problems that were observed in the course appeared to cause real dissatisfaction among the students. Following the Two-Factor Theory [Herzberg 1968; Herzberg et al. 1959], a theory that has remained topical to the present day (e.g., DeShields et al. [2005]; Douglas et al. [2008]; Furnham et al. [2009]), these problems were categorized as *hygiene factors*. The other side of the Two-Factor Theory, the motivators, is referred to in many studies on programming (e.g., Bergin and Reilly [2005]; Guzdial and Soloway [2002]; Jenkins [2001]; Jiau et al. [2009]; McWhorter and O'Connor [2009]; Rodrigo and Baker [2009]), and can further be divided into its intrinsic and extrinsic elements [Lepper and Henderlong 2000]. In this study, not all of the students appeared to have problems in completing the course, so it was concluded that they were driven by an *intrinsic motivation* that helped them through the course in spite of its problems. The study also revealed that, in the absence of tuition fees, some students enrolled habitually in the course without taking any further action to actually complete it. It was concluded that tackling the problems associated with this group of students would require *extrinsic motivators* enforced by the administration. The initial changes in the course decreased the pass rate from 44% to 36% in 2005 but by 2009 the pass rate reached 68%. Together with a positive course atmosphere and receipt of the student union excellence award for its course materials in 2008, evidence of eventual course rehabilitation was provided.

The novelty of this study lies in its holistic approach to improving the results of a first programming course. At present, the literature focuses on individual actions aimed at improving the results of such courses, such as changing the programming language (e.g., Becker [2002]; Böszörményi [1998]; Johnson [1995]; Kelleher and Pausch [2005]; McIver and Conway [1996]; Raadt et al. [2003]; Roberts [1993]), synchronizing curricula and course contents [Plaza and Medrano 2007; Rehman et al. 2009], automatically assessing student assignments [Douce et al. 2005; Higgins et al. 2005], emphasizing the role of process in constructing programs [Boisvert 2009; Caspersen and Kolling 2009], and using visual tools [Kasurinen et al. 2008; Myller et al. 2008]. Some educational institutions have conducted long term efforts to improve the results of CS1 (e.g., Georgia Institute of Technology [Guzdial 2009]), but the literature focuses on course elements rather than the course as a system, which has a serious risk of limiting the improvement efforts to local fitness peaks. Gill and Jones [2010] approach this problem through the concept of a *rugged fitness landscape* [Kauffman 1993]. In a rugged fitness landscape, a dependent variable has a fitness value that should be maximized but, due to the interactions between the independent variables affecting the dependent variable, many variable combinations result in local fitness peaks. After reaching a local fitness peak, any change in the setup leads to a performance drop before the next fitness peak can be achieved. Gill and Jones [2010] study one CS1 course over seven course implementations with changes in the course design and indicate three implementations with high, three with medium, and one with low fitness values based on student instructor evaluations, course attrition, course performance (grades and GPAs), and peer reviews by faculty members. Gill and Jones [2010] summarize that rugged fitness landscapes typically involve hundreds of possible variables which influence different regions, but that new variables are frequently needed to capture new situations, making statistical analysis of limited use. Thus, improving the course results requires a holistic understanding of the course and its elements, and is achieved through a qualitative research approach.

The remainder of this article is structured as follows. The research frameworks used in the study are introduced in Section 2 and Section 3 provides details of the research method used. Section 4 presents the course that is the subject of this study, along with the five years of its implementation and results. Section 5 discusses the findings of the overall study, how they relate to the theoretical frameworks used, and the implications and limitations of the study. Section 6 closes the article with conclusions and future research ideas.

2. RESEARCH FRAMEWORKS

The study uses two conceptual frameworks to analyze and tackle the problems with the troubled course. First, the Theory of Constraints (TOC) was used as an overall systems framework for the course improvement work. Second, since the course success depended upon the motivation of students, motivation theory was used to guide the associated improvement in this area.

2.1 Theory of Constraints

A system consists of two ensembles, the form and its context, from which the form—the system—is the solution to the problem while the context defines the problem [Alexander 1964]. A good fit between the form and context is, however, difficult to observe directly: "we should always expect to see the process of achieving good fit between two entities as a negative process of neutralizing the incongruities, or irritants, or forces, which cause misfit" [Alexander 1964].

The misfit concept has been implemented as a systemic management framework for continuous process improvement and is called the Theory of Constraints [Goldratt 1994]. The Theory of Constraints (TOC) builds on the assumption that the global goal of an organization is to make money: "The goal of a manufacturing organization is to make money, and everything else we do is a means to achieve the goal" [Goldratt and Cox 2004]. The manufacturing process is seen as a sequence of actions that need to be completed to finish the product while constraints are "anything that limits a system from achieving higher performance versus its goal" [Goldratt 1994]. The basic TOC performance metrics are sales (throughput), inventory, and operating expenses [Dettmer 1997].

— *Throughput*. The rate at which the entire system generates money through sales (product or service).
— *Inventory*. All the money the system invests in the things it intends to sell.
— *Operating expenses*. All the money the system spends in turning Inventory into Throughput.

It is claimed that performance improvement can only be realized through increasing the throughput, decreasing the inventory or decreasing the operating expenses. Since zero limits the cutbacks that can be made to the inventory and the operating expenses, increasing the throughput is the best area to focus upon for continuous improvement.

Goldratt [1994] suggests that the improvement process should be defined in two ways using both the terminology of the system to be improved (Table I(a)) and the terminology of the improvement process itself (Table I(b)). Most of the research on the TOC still focuses on the manufacturing industry, even though an increasing number of reports from other fields have been published, such as software development [Anderson 2004], management [Hsu and Sun 2005], project management [Umble and Umble 2000], health care [Lloyd and Lana 2003], and airline services [Polito et al. 2006]. In general, the results have been estimated to generalize to any type of organization [Mabin and Balderstone 2003].

In the educational field, the TOC has been used at a number of levels. At the individual course level, the TOC has been used in developing small-scale case studies to help in Introduction to Information Systems courses [Sirias 2002b], to improve the teaching of business statistics [Sirias 2002a] and strategy formulation [Boyd et al. 2001], to teach problem-solving skills in marketing [Cooper and Loe 2000], and to address ill-structured problems [Walker and Cox 2006]. The improvement of graduation rates is reported by Goldratt and Weiss [2005] who worked with an Israeli school with 200 students in the 9th to 12th grades. At the beginning of the study period in 1998, none of the students managed to achieve a matriculation diploma. However, after the TOC principles were adopted in the school in 1999, the percentage of students missing more than three subjects for the diploma steadily decreased over the subsequent years, while the percentage of students entitled to the diploma increased such that

Table I. The Theory of Constraints in: (a) The Terminology of the System to Be Improved; and (b) the Terminology of the Improvement Process Itself [Goldratt 1994]

The focusing process in the terms of the system to be improved
1. Identify the system's constraints.
2. Decide how to exploit the system's constraints.
3. Subordinate everything else to the above decision.
4. Elevate the system's constraints.
5. If in the previous steps a constraint has been broken, go back to step one, but do not allow inertia to cause a system constraint.

(a)

The focusing process in the terms of the improvement process itself
1. What to change?
Pinpoint the core problems!
2. What to change to? Construct simple, practical solutions!
3. How to cause the change? Induce the appropriate people to invent such solutions!

(b)

67% of the students were entitled to it in 2003. The changes did not require additional resources, only a revision in the way the existing resources were allocated within the system. For example, the subjects were reorganized as larger ensembles to focus on fewer topics at any given time; the exams were allocated evenly throughout the study period rather than just close to the matriculation exam; students took the exam half a year before their last chance to do so, so that they could focus on problem areas during the remaining period if necessary; and 5% of the teaching hours were reallocated as extra resources to handle problems close to the matriculation exams.

In this article and following the TOC, we use the throughput (i.e., the pass rate) as a key measure of the course success. By considering the course as a process with multiple phases represented by the course deliverables (e.g., the exam, course project, and weekly assignments), the bottlenecks constraining the overall pass rate were identified. After constraint detection, the problem areas were studied more closely to identify misfits, and then solutions were sought to eliminate them.

2.2 Motivation Theory

There are a number of theoretical approaches through which the topic of work redesign has been studied, ranging from activation theory through to socio-technical systems theory [Hackman and Oldham 1976]. Attempts to improve the productivity and quality of work experiences are commonly associated with a concomitant interest in the motivation of employees, and there are an equal plethora of theories through which the specific topic of work motivation can be examined, including expectancy theory [Vroom 1964] and Maslow's hierarchy of needs [Maslow 1954]. Although Ambrose and Kulik report that the actual link between motivation and performance has been little studied [Ambrose and Kulik 1999], one influential framework that has been used to investigate the motivation of programmers and software engineers [Hall et al. 2009] is based upon Herzberg's seminal work from the 1950s [Herzberg 1968; Herzberg et al. 1959]. Following empirical study, Herzberg found that the factors involved in producing job satisfaction (and motivation) are separate and distinct from those that lead to job dissatisfaction; satisfaction and dissatisfaction are not opposing feelings. This observation lies at the heart of Herzberg's theory.

Herzberg's Two-Factor Theory (or Motivation-Hygiene Theory) differentiates "motivator factors" from "hygiene factors." Motivator factors are intrinsic to the work that is to be done and the primary cause of satisfaction (e.g., achievement, recognition for achievement, the challenge of the work itself, responsibility, growth, and advancement opportunities). Hygiene factors are extrinsic to the work itself and the primary cause of unhappiness when absent (e.g., company policy and administration, supervision, interpersonal relations, working conditions, salary, status, and security). Hygiene factors are thus also referred to as dissatisfaction avoidance factors.

Herzberg observed that many changes made to the tasks of workers or to the working environment led to short-term improved levels of employee happiness, but rarely resulted in the increased levels of motivation anticipated. Employees acted because some external stimulus had prompted them to act, resulting in a temporary effect on attitudes, which required the continued presence of stimuli to maintain. Such changes effectively introduced hygiene factors into a working context in which they were once absent. They did not lead to increases in motivation, only to less dissatisfaction. To attain longer-term improvements in actual motivation, and to affect longer-lasting positive effects on attitudes, the stimulation for change needed to come from within, from what Herzberg referred to as an employee's own internal generator, and the motivators need to be designed into the work itself.

Whilst directed at employee motivation and of relevance to work redesign, the fundamental ideas of Herzberg's Two-Factor Theory have more general applicability to other settings. Where the objective of any enrichment program is to increase satisfaction and reduce dissatisfaction, the implication is that a composite of separate factors needs to be considered. To reduce dissatisfaction, hygiene factors need to be taken care of systemically and on an ongoing basis. To increase satisfaction, motivator factors need to be introduced systematically.

Motivational psychologists further distinguish between those activities that are intrinsically motivated (i.e., performed voluntarily and without reward) from those that are extrinsically motivated (i.e., performed for an external reward). Consideration of how intrinsic and extrinsic motivation can operate in tandem with one another is a pressing concern for education design [Lepper and Henderlong 2000]. In a professional education setting, Grollman [1974] remarks on how the motivators need to be built into the content of the education, but cautions that "... even if a program is educationally sound in all other respects, its degree of success or failure often hinges on the hygiene factors affecting the total learning environment."

How to design and run a successful first course in computer programming is a subject of much debate within the computer science education community and its associated literature. Much of this discussion revolves around the choice of programming language and whether objects should be taught early or late [e.g., Bruce 2005], or around identifying those factors that are likely predictors of student success on such a course, such as a student's self-efficacy, mental model of programming, intrinsic motivation, math background, learning style, etc. [Bergin and Reilly 2005; Wiedenbeck et al. 2004; Wilson and Shrock 2001]. Such discussion focuses almost exclusively on the content of the course itself and on what Herzberg would refer to as motivator factors. Less attention has been paid to nurturing the wider environment in which such motivators can take hold and within which the content can thrive.

In this article, we study the student motivation to work from three viewpoints. The hygiene factors that have been claimed to cause dissatisfaction and educational program failures are the primary focus of the study. They represent potential misfits in the course arrangements. The intrinsic and extrinsic motivators of the students are also both explored so as to understand their role in course success.

3. RESEARCH METHOD

The case study approach was chosen since it fits those studies where there is limited control over behavioral events and further serves to explain events [Yin 2003]. The study was designed as a multiple-case study with embedded design and literal replication logic predicting similar results, and with the course success criteria, the pass rate, as the primary unit of analysis. Since the boundary with the context is not always self-evident in case studies [Yin 2003], some aspects of the student population, curriculum, department, and university were explored to provide data triangulation. However, the study pays limited attention to course staff, since they were participant-observers in the study itself, resulting in a high risk of biases [Yin 2003]. The individual course implementations over the five-year period represent the five case studies. The research questions under investigation were the following.

(1) Why did the course have such a low pass rate?
(2) How can the pass rate be improved?

The data collected in the study were stored in a case study database [Yin 2003]. For each course implementation, the number of students enrolled and the pass rates were recorded, along with all the deliverables and data on participation in lectures and exercises. Four online surveys were conducted for each course implementation: initial, midterm, final, and dropout surveys. The initial survey focused on demographic and starting level data, while the midterm survey tracked progress, and the final survey focused on the skills after the course and feedback on the course. The dropout survey was conducted after the course among those students who did not complete all of the compulsory assignments. The dropout survey was conducted in 2005 through 2008, but not thereafter, since the number of responses decreased at the same time as the results stabilized. The students submitted all the weekly assignments and projects in the virtual learning environment (VLE) that was used [Viope Solutions Ltd. 2010]. This VLE tested programs for correctness and also provided information on login sessions, submission types, and errors. The prior course results between 2001 and 2004 were acquired from the university's study registry to explore the pass rates prior to the beginning of the study [Pirinen 2008]. Finally, the course's instructional staff documented key events in the lectures and exercises in diaries.

The analysis was conducted in two phases, the first annually after each course and the second after the period of the whole study. Each year, the annual analysis began by developing a longitudinal view on the course enrollments and the pass rates, as well as by looking at the completion percentages of the course deliverables. The quantitative study continued with the survey results and the student submissions in the VLE, then moved gradually to a more qualitative analysis of the student feedback, supported by the observations of the course staff. The data were triangulated from multiple sources and rival explanations were created for the observed problems by multiple persons involved in the course and engaged in teaching in the department. Once plausible explanations for the problems were identified, practical solutions were developed based on the literature and staff experiences. The analysis of the whole study began by forming a list of all the problems observed and all the actions taken during the period of the study. This list was analyzed following the open, axial, and selective coding ideas of Grounded Theory [Strauss and Corbin 1998]: first, the problem categories were identified based on the problems observed in the study; second, the relationships between the categories were explained; and third, the categories were refined to arrive at an integrated set of problems. The present study provides limited cause-effect analysis in the form of statistical analysis for two reasons. First, statistical analysis evaluates the correlation between observed effects and their causes, but these were not all known prior to the study. For example, the study revealed that student motivation should be treated as three separate factors rather than just the one, as hygiene factors, intrinsic motivators, and extrinsic motivators. Thus, due to a limited prior understanding of the problem domain, this study did not build on statistical analysis (cf. Gill and Jones [2010]); rather, it used the case study approach to explain causalities

[Yin 2003] and to identify previously unknown factors impacting the outcome. Second, the course improvement effort was considered ethically more important for the students than finding out the correlations, and thus many improvement actions were undertaken whenever possible; this made estimating the effect of individual actions problematic.

In the spring of 2010, the study reached closure according to the two criteria suggested by Eisenhardt [1989]: “researchers should stop adding cases when theoretical saturation is reached” and “the iteration process [between theory and data] stops when the incremental improvement to theory is minimal.” The definition of theoretical saturation is the one suggested by Glaser and Strauss [1967]: “Theoretical saturation is the point at which incremental learning is minimal because the researchers are observing phenomena seen before.”

The study was documented as a manuscript and validated by the wider departmental staff. All the case study descriptions were first written down in a standard format and causal explanations were developed to identify the entities and the mechanisms that connect them and combine to cause events to occur [Easton 2010]. Rival explanations challenging the results of the study were subsequently developed to consider the collected evidence from different perspectives [Yin 2003]. The case study descriptions were then pruned to focus on the key issues only. The final manuscript was validated from the departmental point of view by three people: the Head of the laboratory responsible for the software engineering curriculum and the present implementation of the course, the Head of the IT Department, and a lecturer who had given an earlier version of the course in the 1990s. The latter two people have been staff in the department since the 1980s.

4. A FIVE-YEAR LONGITUDINAL STUDY

This section introduces the CS1 course that is the subject of the longitudinal study in Section 4.1 and provides a systemic course definition for it in Section 4.2. The replicated case studies are presented in Section 4.3 and a cross-case analysis is provided in Section 4.4. The section closes with a summary of the issues and actions in Section 4.5.

4.1 The First Programming Course

The first programming course, Fundamentals of Programming (CS1), has been a part of the curriculum of the Department of Information Technology (IT) at Lappeenranta University of Technology (LUT) since the establishment of the department in 1986. Starting in 2001, the CS1 course was offered in two variants: course A for IT majors and other students who needed to acquire programming skills; and course B for those students who needed to understand the application of programming without necessarily acquiring actual programming skills (e.g., management students). All the students attended the same lectures and undertook the same programming assignments but, to reduce the course demands, the course B students did not complete a programming project and they did not attend the weekly two-hour teaching assisted lab sessions. Only the course A students were permitted to continue on to those other IT courses that demanded programming skills, such as Data Structures, Design of Algorithms, and Fundamentals of Object Oriented Programming.

The course contents in 2005 covered the basic programming concepts of iteration, branching, data structures, pointers, functions, recursion, error handling, dynamic memory handling, and linked lists in the C programming language. It also covered the fundamentals of problem solving, documentation, and algorithms. The 14-week course consisted of 35 hours of lectures, with 28 additional hours of exercises for the course

Table II. The Elements of a Systemic Course Definition with Their Key Attributes

Course Description	Technical Infrastructure	Teaching	Support	Assessment
Learning objectives	Programming tools	Lectures	Lecture videos	Final exam under supervision
Prerequisites	Study materials	Theory and practice	Teaching assisted open labs	Assignments evaluated weekly
Scope	Course assignments	Processes	Discussion forum	
Philosophy	Policies		Office hours	
			E-mails	
			VLE hint system	

A variant. The student workload was 162 working hours in total for course A and 135 working hours in total for course B, which equaled 6 and 5 European Credit Transfer and Accumulation System (ECTS) credits respectively (1 ECTS credit equals 27 hours of total student effort in Finland [European Commission 2009]). Since the course was aimed at first year Bachelor students, it was given in Finnish only.

4.2 A Systemic Course Definition

The success of an educational program depends on the faculty, the student body, and the infrastructure, along with industry involvement to keep the curriculum relevant and current [Joint Task Force for Computing Curricula 2004]. Since the analysis of the CS1 course under study indicated deficiencies in the infrastructure (Section 4.3.1), the course revision started with a revision of the technical infrastructure (i.e., laboratories and classrooms, literature, and a suite of applicable software tools [Joint Task Force for Computing Curricula 2004]), along with changes in the assessment so as to engage students right from the beginning of the course (Section 4.3.2). The role of the course in the overall curriculum was then revisited in 2007 (Section 4.3.3) and the students were provided with an increasing amount of support material (Sections 4.3.3 and 4.3.4). The teaching itself underwent changes from the beginning of the study period (Sections 4.3.1–4.3.5). Thus, our inductively developed systemic course definition covers all the key areas of the course: course description, technical infrastructure, teaching, support, and assessment (Table II).

A more systematic approach would have been to identify all the variables affecting the course results and attend to each of them, but it would have been impractical to do so. Gill and Jones [2010] report on the use of over a dozen variables associated with course design/delivery alone, which they note as being far from an all-inclusive set, and state that: “No attempt was made to postulate the variables necessary to adequately capture the attributes of instructors, content, instructional technologies, or students.” Consequently, only five elements were identified (cf. [Miller 1956]) to make an interaction analysis between the elements feasible when planning changes.

The course description element serves as an interface between the curriculum and individual courses. Considering a curriculum as a process in which different courses represent steps that focus on specific knowledge and skill areas, the learning objectives of the curriculum can be allocated to individual courses to define the expected knowledge and skills for the students enrolling in a particular course. This top down approach establishes a clear division of goals and responsibilities for different courses, and makes it straightforward to manage the course prerequisites and scopes. Finally, the overall course philosophy was active learning, which we define as instructional activities that encourage students to learn to program by doing programming and thinking about what they are doing (cf. Bonwell and Eison [1991]).

The technical infrastructure element comprises the tools used in the course, such as editors, compilers or interpreters, Virtual Learning Environments (VLEs), programming and debugging processes, the course Web site, and plagiarism detection tools. It also comprises the study materials, such as programming guides, books, and lecture materials. The course assignments include both weekly assignments and a course project, and the policies represent the course rules. The course policies are defined as technical infrastructure to make it possible to adapt them every year, for example, as a response to changes in the environment (cf. Gill and Jones [2010]).

The teaching element focuses on the actual act of educating the students, the process whereby an instructor actively explains the study material to students and demonstrates how tasks can be completed. Teaching can be carried out in many ways but, since the CS1 classes at LUT have more than one hundred students, lecturing is the primary means of instruction. The teaching covers both theory and practice with the aim of providing the students with a model for developing programs and solving problems. In addition to demonstrating the programming process to students, lectures are also used to demonstrate common programming mistakes and ways to solve them.

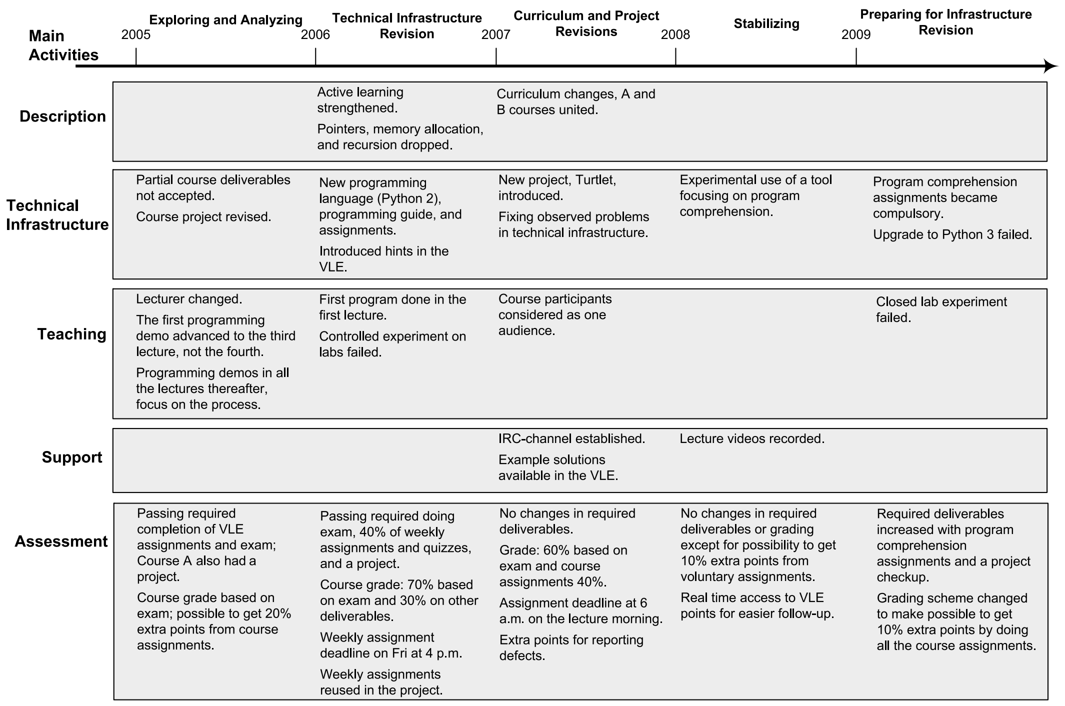
The support element addresses the fact that many students need extra help and practice. In contrast to the teaching element, the students need to take the leading role to benefit from the support element. For example, video recording the lectures and making the videos available allows students to reexamine how the tasks were completed in the lecture. The more traditional forms of support include office hours, e-mail correspondence with students, open labs where students can come and ask questions at their own pace [Thweatt 1994], and different discussion forums to get help from peers or teaching assistants. Additional support is provided via a simple form of intelligent tutoring system (cf. Hume et al. [1996]) where programming hints are provided by the VLE.

Finally, course completion requires passing the assessment element. The learning outcomes are evaluated in the final exam, which forms the basis for course grading, even though the completion of the weekly assignments and the course project are also considered. The final exam is the only deliverable that the students complete under supervision to assure that the grade is representative for that student.

4.3 Replicated Case Studies

The present study started when the instructor teaching the CS1 course at LUT changed in the fall of 2005. Since the course was run once a year thereafter, and the courses were unconnected, the five years that are the focus of the study represent logical replications with the same goal of improving the course success (i.e., the pass rate). Figure 1 summarizes the main changes that were undertaken for each year of the case study focusing on the key elements of the course. Smaller changes were undertaken continuously as and where feasible. In the following subsections, the five case studies are presented chronologically. The subsections follow a similar structure. First, the problems and changes particular to each year are summarized, and then the results of making the changes are reported. The results include participation statistics for every case study, as well as data specific to the individual years that proved important in that study. The mechanisms affecting the course results are hypothesized and the key development needs for improving the course further are presented.

4.3.1 Exploring and Analyzing in 2005. Discussions with the IT department staff at LUT before the course of 2005 brought up two major issues with the prior status of the first programming course. The student programming skills were found to be deficient when



Main Activities	2005 Exploring and Analyzing	2006 Technical Infrastructure Revision	2007 Curriculum and Project Revisions	2008 Stabilizing	2009 Preparing for Infrastructure Revision
Description		Active learning strengthened. Pointers, memory allocation, and recursion dropped.	Curriculum changes, A and B courses united.		
Technical Infrastructure	Partial course deliverables not accepted. Course project revised.	New programming language (Python 2), programming guide, and assignments. Introduced hints in the VLE.	New project, Turtlet, introduced. Fixing observed problems in technical infrastructure.	Experimental use of a tool focusing on program comprehension.	Program comprehension assignments became compulsory. Upgrade to Python 3 failed.
Teaching	Lecturer changed. The first programming demo advanced to the third lecture, not the fourth. Programming demos in all the lectures thereafter, focus on the process.	First program done in the first lecture. Controlled experiment on labs failed.	Course participants considered as one audience.		Closed lab experiment failed.
Support			IRC-channel established. Example solutions available in the VLE.	Lecture videos recorded.	
Assessment	Passing required completion of VLE assignments and exam; Course A also had a project. Course grade based on exam; possible to get 20% extra points from course assignments.	Passing required doing exam, 40% of weekly assignments and quizzes, and a project. Course grade: 70% based on exam and 30% on other deliverables. Weekly assignment deadline on Fri at 4 p.m. Weekly assignments reused in the project.	No changes in required deliverables. Grade: 60% based on exam and course assignments 40%. Assignment deadline at 6 a.m. on the lecture morning. Extra points for reporting defects.	No changes in required deliverables or grading except for possibility to get 10% extra points from voluntary assignments. Real time access to VLE points for easier follow-up.	Required deliverables increased with program comprehension assignments and a project checkup. Grading scheme changed to make possible to get 10% extra points by doing all the course assignments.

Fig. 1. The course timeline for the study period, indicating the main changes to the course elements each year.

it came to the follow-on courses and the students participating in the course were felt to have immature study practices. As examples of immaturity, the students returned coursework from previous years at any time during subsequent years and expected that they would be evaluated and graded immediately, the student participation in the lectures and exercises was limited, and the students expected solutions to the exercises rather than help in the attainment of these solutions. Due to the focus on exploration and analysis of the course problems in 2005, only two changes were undertaken to the course implementation itself. The course A variant introduced a new and simpler project, focusing on file and string operations without linked lists, and a policy of not accepting partial coursework from previous year courses was enforced.

A new instructor taught the course in 2005, while two of the three teaching assistants remained constant from the prior year. As in the previous year, the course included 35 hours of lectures based on 347 lecture slides compiled in Finnish by the previous lecturer, and the students had to submit 30 small C programs in the Virtual Learning Environment (VLE). The role of programming in the lectures was increased, however, by advancing the first programming demonstration to the third lecture and by going through two to five programming demonstrations in each lecture. The course assessment was based on the exam result, but students could get 20% extra points by completing the VLE assignments to schedule during the course.

Only 48% of the 249 students enrolled in 2005 completed all the course assignments and only 36% got a grade. This was an 8% unit drop from the previous year. A dropout survey conducted among the failed students (Table III) revealed that the four most common reasons for dropping the course were schedule conflicts, course requirements, required amount of time, and motivational problems. Based on the qualitative feedback from the final survey sent to all the students enrolled in the courses (n = 77),

Table III. Reasons to Drop the Course in 2005

The online dropout survey was sent to 131 students and 53 (40%) responded to the survey. The students were asked to estimate to what degree the named factors contributed to their dropping the course using a Likert scale of 1 to 5, 1 being strongly disagree and 5 being strongly agree. The table shows the average responses for each factor.

Factor	**Average**
Due to schedule conflicts I could not participate in the lectures and/or do the required assignments.	3.4
I failed to motivate myself enough to complete the course.	2.9
Completing the course seemed to require more time than I was willing to invest in it.	2.8
The course requirements proved too difficult for me.	2.8
The course proved too laborious in comparison with the credit units available.	2.4
The course did not prove as interesting as I assumed based on the available information.	2.0
Sickness or other personal life event made it impossible for me to complete the course.	1.9
The teaching style and methods did not fit me.	1.9
I did not get help when I needed it.	1.7
I did not understand what the course was about.	1.5

the most difficult part of course A was reported to be the course project, while the course B students found the programming questions difficult on the exam, reporting that their overall programming skills were limited. The changes made in 2005 did not improve the course success.

The data collected did not readily indicate the root causes of the low pass rate so we had to hypothesize the mechanisms [Easton 2010] behind it. The schedule conflicts (Table III) are a contingent issue as the study office of the university constructs the timetables such that mandatory courses do not overlap for students taking the courses during the year of study proposed in the study guide, which is year one for the first programming course. However, since 60% of the students had started their studies before 2005, the study office did not account for their schedules. Many of the other reasons for dropouts were related to motivation (motivational problems, not interesting), workload (required amount of time, not enough ECTS credit) or complexity (course requirements). The final survey data (n = 77), however, suggested that the workload was not a real issue—42 of the course A students averaged 85 hours of work, with a standard deviation of 44 hours and a maximum of 213 hours, while the expected workload for the course was 162 hours. The majority of the students were, in reality, working fewer hours than the course demanded of them. The course complexity was determined to be a real issue. Teaching two course variants with different audiences in the same lectures made it problematic to achieve the varying course objectives (cf. Bills and Canosa [2007]; Forte and Guzdial [2005]; Kinnunen and Malmi [2008]). Many of the students got tired of the lectures trying to serve both audiences and they stopped coming to the lectures. The study materials were also complex, comprising lecture slides, VLE study materials, and any other material found in libraries and on the Internet. In the absence of a single reference book for the course, all the course topics had to be covered in the lectures, resulting in a large package of slides. The slides were not designed to account for pedagogical issues associated with self-directed learning, such as thoroughly thought through example programs. The technical arrangements were also challenging for the students, since programming was undertaken in the UNIX environment from Windows workstations using the C programming language. Some students used C compilers that they found on the Internet and different default compilation arguments led to different program behavior on the PCs and in the VLE that used a HP-UNIX compiler. Observation of the students made it evident that advanced topics such as memory allocation, pointers, and linked lists were difficult for many of them. Further, the students started the course with active participation, but this diminished as the course progressed. Only during the last few days before the final course deadlines did many of the students attempt to do all the course assignments, as demonstrated by the following student comment (the course ended on Dec 9th).

> “I have a bad habit of getting a good enough work morale of myself to complete larger course projects only when necessary—roughly two weeks before the deadline. Consequently, I missed the original project deadline since I failed to command myself in the extraordinary world of the C-language in the extent required in November.”

Overall, the analysis suggested that the students enrolled in the course because it was mandated by the curriculum, but the course itself did not establish clear steps to help the students complete it. This meant that the same students enrolled in the course year after year. It also suggested that the students had problems focusing on learning how to program as they were frustrated by all the surrounding complexities, such as the two course variants with its two goals and audiences, the scattered study materials, and the technically complex programming environment. The Two-Factor Theory refers to those problems that are extrinsic to the work itself as hygiene problems.

4.3.2 Technical Infrastructure Revision in 2006. The starting point for the course in 2006 was the perception that, in the attempt to satisfy the requirements for different sets of stakeholders, the course was distracted from what should be its primary goal to teach the students to program. Consequently, an attempt was made to revitalize the course by (1) making a departmental decision to unify the two course variants in 2007, (2) starting efforts to eliminate the hygiene problems to promote a student’s ability to achieve the course objectives, and (3) increasing the role of doing programming in the course.

The ensuing changes covered all of the course elements in 2006. The course description was adapted by dropping advanced topics like pointers, memory allocation, and recursion from the course content, and an active learning philosophy—doing programming and thinking about it (Section 4.2)—was emphasized. At the same time, the overall conformance to the Computing Curricula 2001 was confirmed [Joint Task Force for Computing Curricula 2001]. A new technical infrastructure was built around the programming language of Python [Python Software Foundation 2010], which was found simpler and easier to learn than the previously used C programming language [Kasurinen and Nikula 2007]. However, the language change required a rewrite of all the course assignments. The number of assignments was increased by a third to 40, and they were complemented with quizzes. A Finnish programming guide for Python was developed as the course study material [Kasurinen 2008]. Since the VLE did not have a module with study materials for Python, a hint system was developed for it. The teaching itself changed little, except that the first program—print “Hello world”—was discussed in the first lecture. The weekly lab-based exercises were modified to allocate students to groups. The course assessment was changed so that 70% of the final grade came from the exam points and 30% came from the completed weekly assignments. The weekly assignments had to be submitted every Friday by 4 p.m., and some were designed to be reusable in the course project to encourage their completion.

Two hundred nineteen students enrolled in the course in 2006 and four teaching assistants ran the labs, two of which were new. The course was completed by 54% of the students indicating an 18% unit improvement in the pass rate. Based on the dropout survey, the most common reason for dropouts in the A course was now motivational problems (n = 34), but no significant changes in the underlying problems were

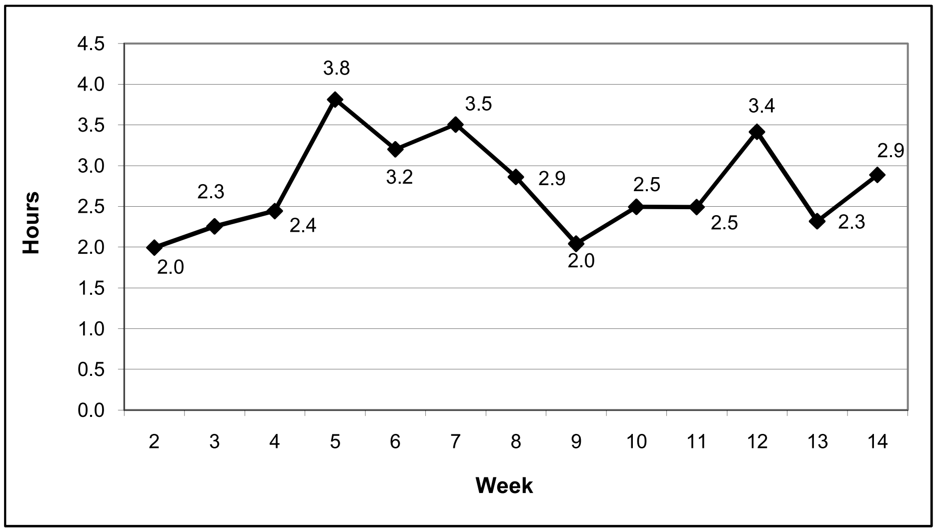


Fig. 2. Student weekly effort on the course in the VLE, based on the system login and logout data.

observed. The biggest issues in the course appeared to be the project and the lack of example solutions to the weekly assignments, but many students were also irritated by the typos in the new course materials. A number of students also reported that they found the VLE system unfair since they could not return their fixed submission a few minutes after the deadline, Friday at 4:00 p.m.

Even though no primary data were collected to explain the course pass rate improvement, we hypothesized the mechanisms as course elements that now supported student learning and the passing of the course. Students now had weekly assignments to complete for assessment purposes, so they had started working on the course earlier than before. Since, on average, only 42% of the students attended the lectures in 2006, the only way that they could have achieved the course learning outcomes was through study of the lecture material and the supporting study materials. Following student frustration at closed group labs, the labs were changed to open labs so that students could come to ask for help as and when needed, rather than follow an a priori determined agenda [Soh et al. 2005]. Since the project had proven a problem in the previous years, some parts of the project were first implemented by way of the weekly assignments, so students gained a smoother and earlier start with the project. By advancing the first programming demonstration to the first lecture, the main tool of the course was presented when the majority of the students, 73%, were present. The active programming in the lectures continued throughout the course and a simple debugging session was included to demonstrate a troubleshooting process. The students with good visual memory could recall how things were done in the lectures and others could re-access the programs developed in the lectures from the course Web page as needed.

The analysis of the collected course data did not indicate any new major issues. Based on the login data from the VLE, the weekly average time the students spent in the VLE during the course varied between 2 to 4 hours (Figure 2). The average effort for the course (n = 125) increased from 85 hours to 87 hours, while the median stayed the same (80 hours), and the standard deviation increased from 44 to 52 hours. Since the student performance in the previous course varied so much, a question on the prior programming experience of the students was added to the initial survey.

Based on the survey (n = 159), 42% of the students could not program prior to the course, 35% knew some basics of programming, and 23% of the students had used programming languages before. This result confirmed that most of the students were new to programming and that the course was correctly positioned as the first programming course.

The biggest problem appeared to be the lack of motivation to complete the course project, even though feedback indicated that the course was doable provided the student just made a serious effort to study the materials. Thus, eliminating the irritating hygiene factors and improving student motivation to complete the course were judged as the most important improvement areas to focus on when moving forward.

4.3.3 Curriculum and Project Revisions in 2007. In 2007, the departmental curriculum was updated by introducing a new course sequence comprising the unified CS1 course in the fall term of the first year, followed by a new practical programming course with the C programming language in the spring term, then a revised course on Data Structures and Algorithms in the second year. Each was standardized as a 5 ECTS course. The changes to the CS1 course focused on the course project and on eliminating as many of the small hygiene problems as possible.

The technical infrastructure was revised by introducing a new course project and by fixing problems in the assignments, programming guide, and lecture materials. The new project, Turtlet, aimed at motivating students with a graphical user interface and a turtle figure [Kasurinen et al. 2008]. At the beginning of the course, the students received a fully functional application without source code and then, during the course, they replaced the existing functionality with their own code. The teaching continued to evolve, targeted at one audience, defined as Bachelor students who could benefit from programming skills in their studies and careers. As support actions, the solutions to the course assignments were available for all the students in the VLE after the submission deadline, and an Internet Relay Chat (IRC) channel was established to facilitate support from peers and teaching assistants. The assessment was changed to reduce the role of the exam to 60% of the course grade, while the other assignments accounted for 40%. Extra points were awarded to the first student reporting defects in a weekly assignment, to motivate the students to search and report on problems, as well as to improve the quality of the assignments themselves. Finally, the previous year's VLE login data (Figure 3) indicated that few students were online at 6 a.m., so the submission deadline was changed for all the weekly deliverables to 6 a.m. This way, most of the students were expected to have stopped working on their assignments well before the deadline. The previous deadline of 4 p.m. coincided with the favored time of most students to actually work on their assignments, so it led to student dissatisfaction.

One hundred sixty-one students enrolled in the course in 2007 and two teaching assistants with previous course experience ran the labs. The course was completed by 65% of the students indicating a 12% unit improvement in the pass rate. However, even though the pass rate had improved, the learning outcomes were not impressive. Baker et al. [2008] proposed a Bayesian Knowledge Tracing algorithm that can be used to estimate programming skills, and we analyzed 658 suitable source files from our VLE to estimate how well the students had learned some of the key concepts from the course [Kasurinen and Nikula 2009]. The percentage of the students that were estimated to have learned the concepts varied between 46% and 67% (Figure 7, Section 4.4). In 2006, the course project was completed by 81% of the students who completed all the weekly assignments, while in 2007 this figure was 92%. Furthermore, 21% (23) of the students completed the project in 2007, even though they did not complete the required amount of weekly assignments. The discussion channel received divided

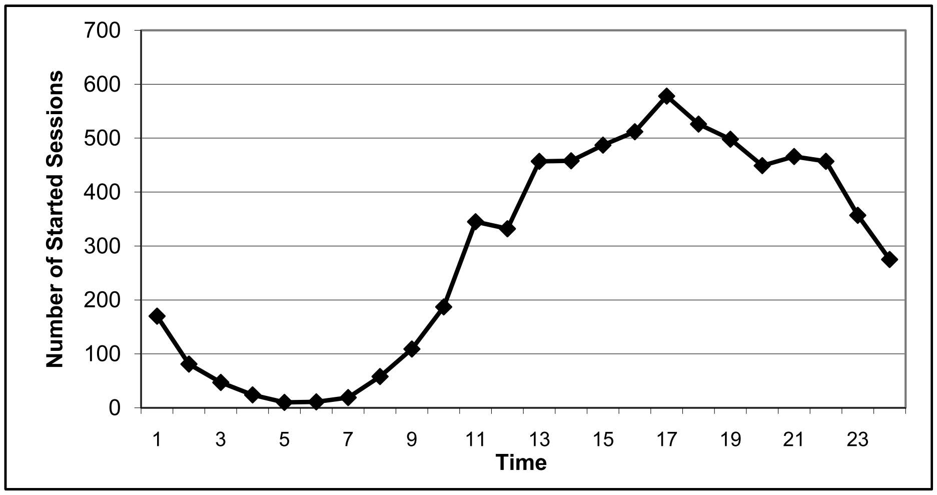


Fig. 3. The VLE system recorded each student session by the login time and duration. Since the average duration of the sessions was 28 minutes, the intensity of the system use can be estimated by looking at the number of sessions started at different times around the clock during the course—the system was used the most at 5 p.m. and used the least at 6 a.m.

acceptance, reported as very useful by 12% of the students ($n = 90$), whereas 71% of the students had not used it at all.

The changes made in 2007 had few directly visible consequences, so the mechanisms of interest focused on the absence of misfits. In 2007, 88% of the students passing the course responded to the final survey while a year earlier, with all the major changes, some students who did not pass the course responded to the survey, resulting in a response rate of 109%. Equally in 2007, the number of qualitative comments decreased by 42% and their total word count by 63%. Only 25% (14) of the failed students returned the dropout survey, yielding no new insights into issues. In general, 39% of the passing students gave feedback on the course from which about two thirds were positive including, for example, acknowledgements for giving points for defect reports and the transition to less mathematical assignments. The new Turtlet project also received positive comments, such as "coding is much nicer when the system visualizes the effects of the commands," but it was also found to "limit the creativity," and some students expressed that they would have rather done some "real programming." The varying perceptions of the course were demonstrated by the following two student comments in the final survey.

> "This is definitely one of the hardest and most annoying courses since the assignments had to be returned every week. I missed some weekly assignments when I was on holidays since the time for doing the assignments was so short."

> "Well, the fire and brimstone like feedback from the last year will not repeat itself anymore. In the beginning I copied most of my programs from the last year, but when I started to read up on it and continued to work on the assignments, I even started to understand what was going on there."

Overall, we hypothesized that those students not taking the course at the university prescribed time had two major problems: course scheduling issues that prevented lecture attendance and prior negative experiences of the pre-Python version of the course. For students in general, the main problems appeared to be the inaccuracies in the assignments and the increasing number of individual preferences in the way the course was run, as evident in the comments above. However, the simpler one-course structure started to ease the implementation and administration of the course for the staff.

4.3.4 Stabilizing in 2008. Faced by the prospect of reductions in course resources, coupled with the teaching assistant who had implemented most of the changes in the technical infrastructure in the past two years moving to a new project, no major changes were undertaken in 2008. The biggest change was starting to record and publish videos of the lectures. Since 2005, the role of coding in the lectures had steadily increased and, by 2008, most of the lecture time was spent writing and discussing code. This was shown on the lecturer's laptop. Everything the students saw on the projector and heard in the lecture hall was also captured on video and released unedited on the course Web page. Another change was the introduction of additional voluntary assignments, following student requests. A visual learning tool was adopted [Rajala et al. 2009] in the tenth lecture of the course and the students could get extra points for their course grade by doing the associated assignments. This change was done on an experimental basis and the assignments did not replace compulsory assignments. A third change concerned the course assessment, which was simplified by adding a real-time report of the earned points and cumulative point gains in the VLE.

One hundred seventy-five students enrolled in the course in 2008 and two teaching assistants ran the labs, one with previous experience of the course. The course was completed by 61% of the students, indicating a 4% unit decrease in the pass rate. On the positive side, 47% of the students got extra points from the voluntary assignments and many students praised the lecture videos. The perception that the course was improving was further supported by receipt of the student union excellence award for its course materials.

In 2008, the student motivation to complete the course assignments caused concern. During the course, 133 students received sufficient points to pass the weekly quizzes and 129 students passed the weekly assignments, but only 126 of them passed both. The project was completed by 114 students, but only after two deadline extensions. Four days before the original deadline, 54 students had returned the project and, by the deadline itself, only 79 projects were returned. Due to a misconception among the course staff, a two-week extension was announced two days before the original deadline, which may explain the low count of projects by the original deadline. By the new deadline, 101 projects were returned. Another extension of the deadline by one week resulted in additional submissions and 114 projects were finally returned acceptably. Even though the number of accepted projects increased by 7 from 2007, the change was not statistically significant ($p = 0.10$) using the two-tailed two-proportion z-test and could simply be due to chance. However, the late start on doing the project, as well as the limited interest in completing all the compulsory assignments, was striking—only 40% of the students who returned the project acceptably after the original deadline contacted any of the teaching staff about it, even though they were instructed to do so. Overall, it seemed like completing all the compulsory assignments was not very important to the students and, from the course management point of view, this became a serious problem.

The usefulness of different course components was studied in the final survey. The students rated the usefulness of each of the 13 items with a 5-point Likert scale from 1 (totally useless) to 5 (very useful), including a Not Applicable option. Ordering the responses by the average usefulness, in decreasing order, produces the following list: programming assignments, programming guide, personal help, exercise sessions, programming examples, lecture videos, lectures, lecture slides, course project, quizzes, optional assignments, hints in the VLE, and the discussion forum. This supports our initial perception of the importance of the active learning philosophy, together with clear study materials.

Overall, it appeared that the course had resolved most of the hygiene problems and that students were fairly satisfied with the course. From the student point of view, the main problem was still the inaccuracies in the weekly assignments. From the staff point of view, the project needed a more systematic implementation because permitting deadline extensions to encourage submissions was not ideal, and signs of plagiarism brought concern.

4.3.5 Preparing for Infrastructure Revision in 2009. The announcement of Python 3 at the end of 2008 created a need to upgrade the course tools, since the course philosophy was to use the newest available stable release of a language. However, the upgrade was postponed because Turtlet used libraries that had not yet been ported to version 3. Thus, the main change in the technical infrastructure was adopting the previously tested program comprehension tool as a standard tool within the course, plus a requirement to complete 40% of the associated assignments to pass the course. In the teaching element, the last lab session of the week was announced as a closed lab to offer extra teaching to the students struggling with the assignments. The course assessment was changed so that the students could get 10% extra points by doing all the course assignments and an initial version of the project had to be presented in an intermediate project checkup three weeks before the final submission.

One hundred fifty-seven students enrolled the course in 2009 and two teaching assistants ran the labs, one with previous experience of the course. The course was completed by 68% of the students indicating a 7% unit increase in the pass rate. The programming knowledge estimation study was replicated by following the procedure developed in 2007 (Section 4.3.3), which showed a 14% unit improvement on average in the results (Figure 7 in Section 4.4). 111 students passed the intermediate project checkup and also completed the project. The student grade average increased 10% from 3.14 to 3.46 and the standard deviation decreased from 1.29 to 1.26. However, even though the grade distribution now resembled a more normal distribution than before, the grading system still left room for improvement. Since the dropout survey had stopped providing insights, it was not issued in 2009. Also, the feedback from the final survey (n = 100) had become positive and less useful for ongoing course improvement. For example, 13 of the 77 students (17%) suggesting development needs noted the time of the lecture, Monday morning at 8 a.m., as a problem. Even though the changes had removed much of the accidental complexity of the course, it still contained a fair amount of essential complexity reflected by the number and nature of the topics covered (Table IV).

Analyzing the mechanisms affecting the course results in 2009, the most interesting revolved around the intermediate project checkup. 81% of the submissions were completed unaided, 9% were completed after some help from an assistant, and the last 10% were completed within a planned three-day extension and with some extra help. All the students passing this intermediate checkup also passed the final project. The possibility to get extra points resulted in improved course results, which was also noted in the final survey as a motivating factor. The closed lab experiment failed, however, since the students who came to this session announced that they did not want the closed format, just hints to help solve their problems. Despite the positive feedback on the program comprehension assignments in 2008, the feedback in the 2009 final survey was not very positive, and the students reported that the assignments were more annoying than motivating.

Table IV. The Main Contents of the Course by the End of the Study in 2009

Program execution	**Data structures**	**Processes**	**Design alternatives**	**Architecture**	**Algorithms**	**Tools**
Sequential	Variables	Development	Files: text binary	Modules	Steps	Python language
Branching	Data types	Testing	User interface: char vs. graphics	Interfaces	Program efficiency	IDLE editor
Loops	Dynamic structures	Debugging	Development: agile vs. plan driven			Compiler/ interpreter
Functions	Objects	Troubleshooting	Programs: imperative vs. object oriented			Virtual learning environment
Exceptions	Files					Debugger

From the student point of view, the course brought up problems associated with managing the many kinds of assignments and tools used in the course. From the staff point of view, the most important problem was the need to upgrade to Python 3 to comply with the course philosophy of keeping up-to-date with technology. The minor issues included reconsideration of the intermediate project checkup timing, since 21% fewer weekly assignments were returned in the week of the checkup in 2009 than in 2008. Also, the grading system appeared too generous, so tuning it more toward the normal distribution appeared justified.

4.4 Cross Case Analysis

Every course implementation since the fall of 2005 was followed by a final survey. Two questions in the survey asked the students to grade the teaching methods used and the course as a whole using a Likert scale from one to five. The teaching methods received an average grade of 3.64 in 2005 and 4.02 in 2009, while the overall course grade was 3.34 and 3.86 respectively. During the five-year study period, the study methods improved 11% and the overall grade improved 16%.

Figure 4 shows the student enrollments in the course between 2002 and 2009, together with the course pass rates. Based on the study registry data, the course had more than 400 students enrolled prior to 2005, but only 44% to 47% of them passed the course annually. In 2005, along with the lecturer change and the changes in the course itself, the pass rate dropped to 36% at the same time as the student enrollments dropped by 44%. Another larger change in the course enrollments took place in 2007 when the number of enrollments dropped by 26% from 219 to 161. After a statistically significant improvement in the pass rate in 2006 and 2007 ($z = -3.91$, $p = 0.001$ and $z = -2.15$, $p = 0.05$ respectively), the improvement dipped in 2008 from 65% to 61% and reached 68% in 2009, without the changes being statistically significant.

The larger drops in the enrollments in 2005 and 2007 (Figure 4) suggest curricula level changes in the role of the course, since the percentage of students entering the university dropped only 6% and 4% respectively. An analysis of administrative events revealed that, up until 2004, all the universities in Finland offered only five-year curricula leading to a Masters degree but, to comply with the Bologna treaty [European Commission 2010], two-level degree curricula with both Bachelor and Masters degrees were adopted in all the Finnish universities in the fall of 2005. In January 2007, the LUT organization was itself revised by introducing another layer in the organizational structure, referred to as faculties, and the departments became administered through these faculties. As a result of both of these contingent events, all the curricula in the university were revised and the role of CS1 was altered in many of them at the same time.

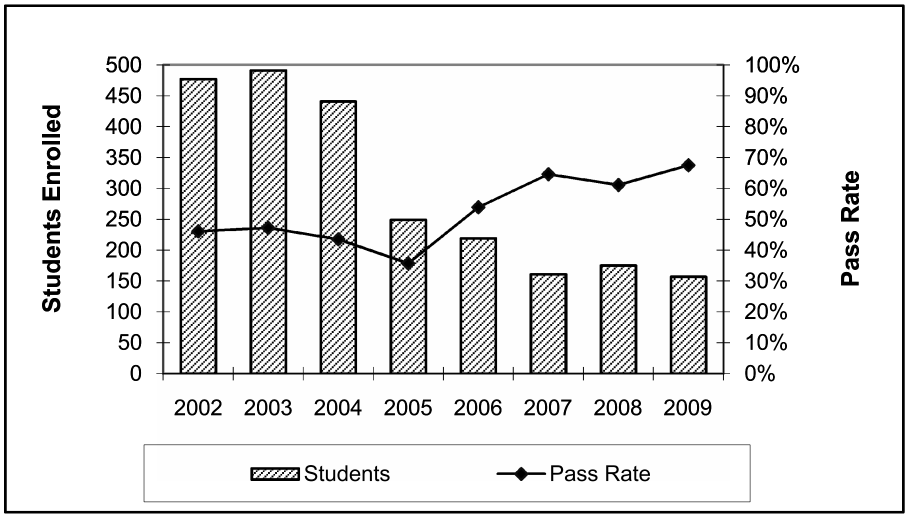


Fig. 4. Student enrollments and pass rates between 2002 and 2009.

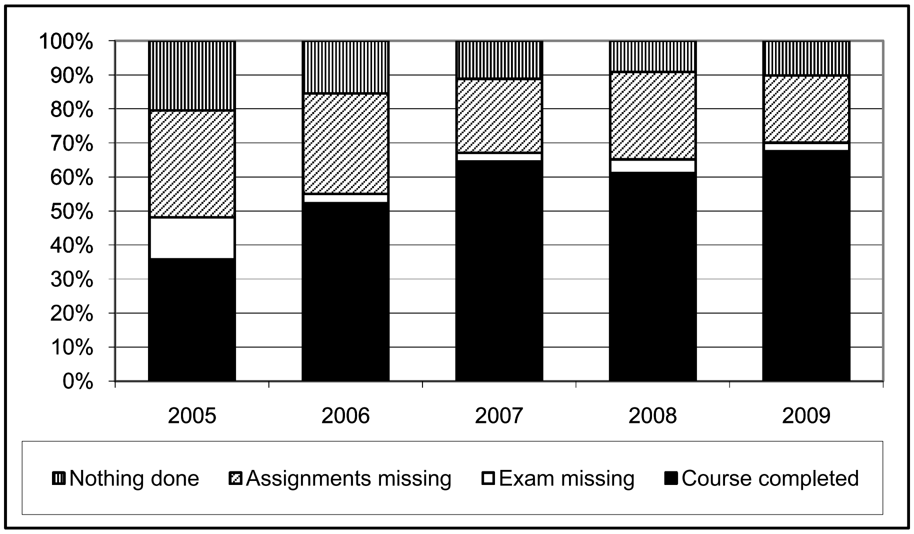


Fig. 5. The percentage of students completing the various course elements. The *Exam missing* category refers to students that miss only the exam from the course completion, while the *Assignments missing* category refers to students who did not complete all the assignments and need to enroll in the course again.

Starting in 2005, detailed data were collected regarding the course results and the completion of the course assignments (Figure 5). After the major course revision in 2006, the percentage of students who missed only the exam had been below 4%. The fact that 21% to 9% of the students did nothing but enroll in the course could be attributed to the fact that university education is free in Finland and, consequently, this behavior is somewhat prevalent. The biggest issue for the course was that some students completed only a portion of the course assignments. This group of students varied between 31% and 20%, and continues to be the main challenge for the course improvement today.

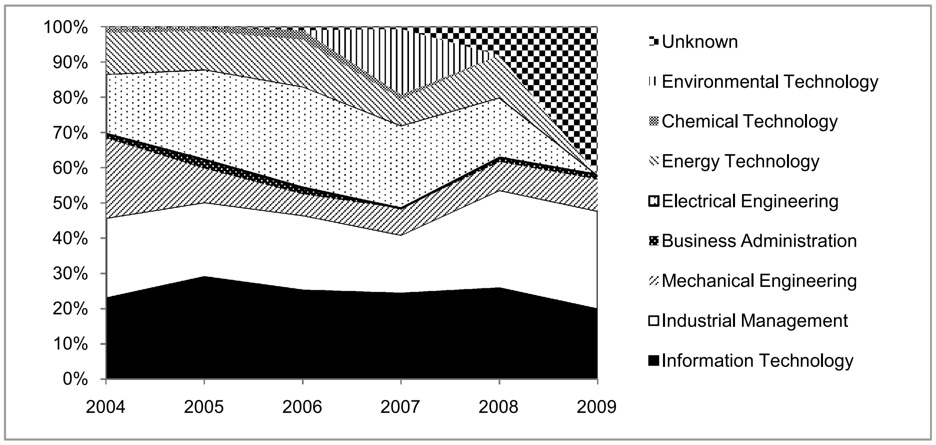


Fig. 6. The proportion of the different majors of the enrolled students, before and during the study.

In an attempt to clarify the reason for the changes in the results, data were collected on the students' majors (Figure 6). The greatest change took place in 2005 when the student enrollment in the course fell from 441 to 249. Even though the changes were large for some majors (e.g., the number of Mechanical Engineering students fell by 76 (76%) and the number of Industrial Management students fell by 48 (48%)), the changes occurred fairly evenly across all the enrolled majors and the proportion of different majors did not vary substantially. The greatest change in the pass rate occurred in 2006 when there was less change to the overall distribution and numbers of the various majors. In 2007, the distribution of majors was temporally disturbed since all the Environmental Technology majors were required to take this course. Finally, in 2009 the major of 66 students was not known since this information was no longer readily available for the students majoring in Energy Technology, Environmental Technology, and Electrical Engineering.

The programming knowledge of the students was first estimated with the Bayesian Knowledge Tracing algorithm in 2007 [Kasurinen and Nikula 2009], and the study was replicated for the course of 2009, based on 658 and 651 submissions from the VLE from 120 and 109 students respectively. Both of the estimates are shown in Figure 7, which indicates that all the studied constructs had been learned by 80% to 91% of the students ($p = 0.05$). The figure also shows that the results had improved by 10% to 26% units in the 2009 study. The results have not yet been studied in depth, so there is no definitive explanation for the improvement, but the outcome suggests that the course results are also improving with regard to the learning of the program constructs since the studied constructs have not received any special attention in the course improvement efforts. While the instructors of the follow-on courses reported deficiencies in their students' programming skills prior to the study (Section 1), only anecdotes of a positive gut feeling were provided thereafter and no systematic studies on the topic were conducted. The absence of misfits is not always as easy to observe as their presence (Section 2.1).

In the ideal situation, all the students would take the course only once and as scheduled. However, since many students enroll in the course multiple times, one interesting performance metric is the percentage of students enrolling in the course for the first time (Figure 8). We therefore gathered data on the number of previous registrations by the students as part of the initial survey on the course. As a result of the low pass rate in 2005, only 41% of the students participated in the course for the first time

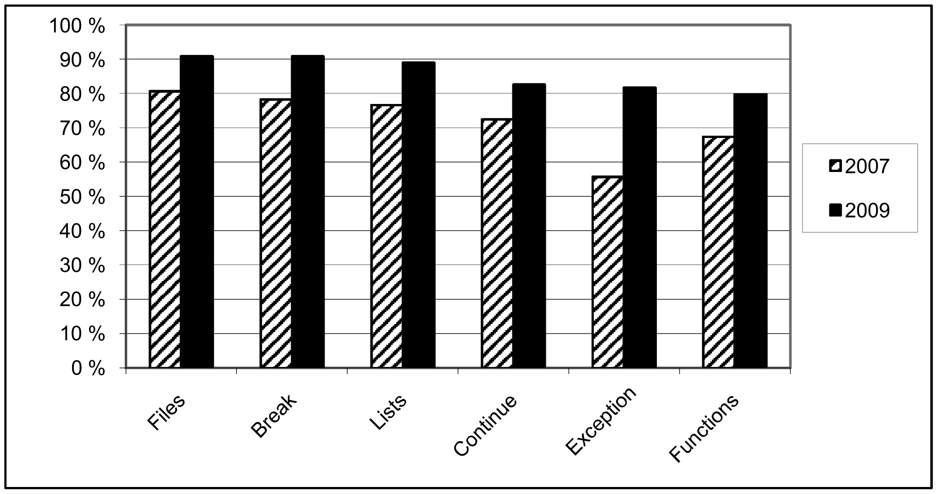


Fig. 7. The estimated programming knowledge in 2007 and 2009. The programming constructs used in this exploratory study were those that had data suitable for the analysis procedure [Kasurinen and Nikula 2009] and do not reflect our view of the most important programming constructs. The *Files* category refers to file handling (open and close), *Break* and *Continue* refer to exiting from a loop, *Lists* refers to a dynamic data structure, *Exceptions* refers to error handling, and *Functions* refers to the use of subroutines (calls and parameters). The analysis estimates the probability of learning the construct of interest based on the pattern of correct versus incorrect use in the submissions [Baker et al. 2008].

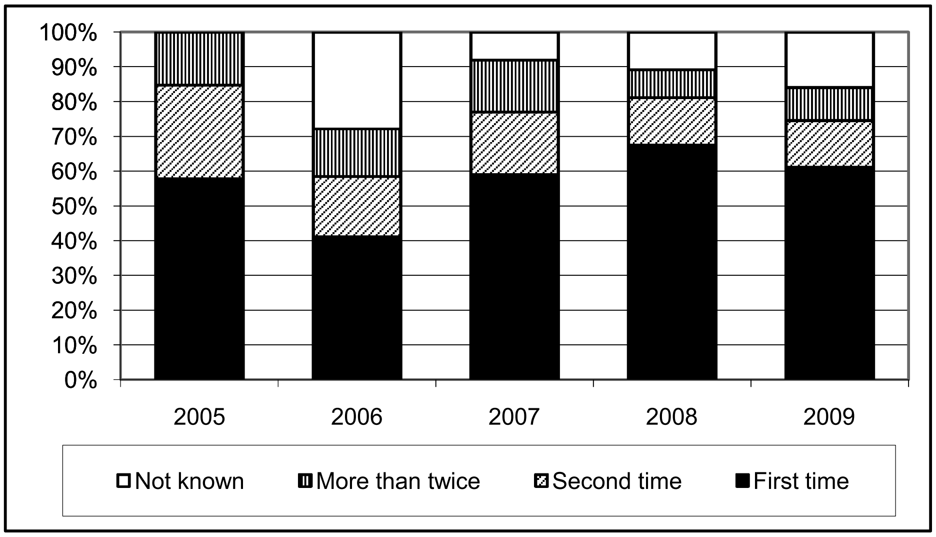


Fig. 8. Number of course enrollments.

in 2006, while the figure was 67% in 2008. The percentage of students who enrolled in the course more than twice also reduced between 2005 and 2009, from 15% to 10%.

4.5 Issue and Action Summary

Table V summarizes the main problems addressed by the course improvement work during the study. As explained in Section 4.3, each course description in Sections 4.3.1–4.3.5 starts with a description of the problems being tackled and the changes being made, and closes with a summary of the further development needs identified. The identified needs do not always match the actions taken for the consecutive courses since the tackling of some problems required background work and were thus addressed over multiple years. This applied to the confusion that resulted from running two course variants (Problem 2.1 in Table V) and the required change in the programming language (Problems 2.3 and 2.6 in Table V). Also, tackling some problems required additional resources, so these could only be implemented when resources became available. In particular, a research assistant was hired in the spring of 2006 to develop the Python course materials (Problems 2.3 and 2.6 in Table V) as a Bachelor thesis [Kasurinen 2006], and another undergraduate student implemented the new project with a graphical user interface (Problem 1.3 in Table V) as a Bachelor thesis in the summer of 2007 [Pirinen 2008]. Therefore, the action selection was guided by the perceived importance of the problems and the available resources (cf. triage [Davis 2003]); major changes, such as offering genuinely different courses for different majors (cf. [Guzdial 2009]), were not considered economically feasible from the outset.

Table V shows three problem categories: student related, course complexity, and general problems. The study suggests that many students had limited intrinsic motivation towards introductory programming and completing the course, and they also had immature study practices.

The course complexity in Table V is described as four subtopics: goal, substance, implementation, and technical complexities. The goal complexity refers to multiple and/or unclear course goals, and the substance complexity reflects the fact that programming involves a number of topics that are non trivial. The technical complexity cannot be avoided totally in a practical discipline such as programming while the implementation complexity is, in many cases, accidental and avoidable.

The general problems covered contextual issues such as staff resource limitations, university control systems, and technological development in the problem domain. Changing a course and increasing the number of student assignments required extra resources, but a course revision should, in the long run, aim at optimizing the resource needs and uses. The study demonstrates that directing the student behavior with extrinsic motivators requires a holistic control system to be effective. In particular, some students did not seem to have any pressing reason to complete the course before graduation, since that was the only time its completion was checked for—not, for example, when they enrolled in a new course with CS1 as a prerequisite. Finally, technical progress cannot be halted, but there is a need to adapt to available tools and technologies.

5. DISCUSSION

This section of the article discusses the research questions we originally posed (Section 5.1), the lessons learned (Section 5.2), their implications (Section 5.3), and the limitations of the study (Section 5.4).

5.1 Research Questions

The research questions of the study are discussed in Sections 5.1.1 and 5.1.2, and the key rival explanations are explored in Section 5.1.3.

5.1.1 Why Did the Course Have Such a Low Pass Rate? The low pass rate for the studied course resulted mainly from three factors: programming as a discipline, course arrangements, and student behavior. First, learning to program is not easy. This is evidenced by a wealth of literature since the 1970s. The present study supports the perception that programming bears a certain amount of essential complexity and learning to program requires a conscious effort. Second, the course arrangements

Table V. Observed Problems in the Case Studies with Problem ID, Description, Category, Summary of Actions Taken, Year(s) Observed, and Year(s) Acted Upon

The IDs group the problems in three groups: student-related problems (1.n), course-related problems (2.n), and general problems (3.n). The issues are ordered by the three groups and by the problem categories.

ID	Problem Description	Problem Category	Actions Taken	Observed	Acted Upon
1.1	Students returned course projects when it suited them, without adherence to the given deadlines.	Immature study practices	Enforced a policy of accepting deliverables only by the given deadlines.	2005	2005
1.2	Students worked only when deadlines approached.	Immature study practices	Introduced weekly quizzes and assignments in 2006, with a requirement to complete at least 40% of them; introduced intermediate project checkup in 2009.	2005	2006 2009
1.3	Students lacked of a serious desire to complete the course, demonstrated by the fact that only few students utilized the deadline extensions offered.	Lack of motivation	Introduced a project with a graphical user interface and a turtle figure; initial parts of the project were done as weekly assignments.	2005 2008	2007
1.4	Students had other lectures at the same time as this course, since the study office checked only the freshmen for schedule conflicts.	Lack of motivation Implementation complexity	Developed study material (see 2.10 below) and started recording the lectures to make off-line listening and viewing possible (see 2.4).	2005	2006 2008
2.1	Two course variants with different learning objectives shared the lectures and most of the assignments.	Goal complexity	Eliminated the course variants and introduced two courses with clearly sequential goals: one for basic programming concepts and a follow-up course for advanced programming concepts.	2005	2007
2.2	The course used the C programming language, since the IT faculty required that the students know it when they start the second year studies.	Goal complexity	Introduced a new course focusing on the C-language for the first year spring term (see 2.1).	2005	2007
2.3	Course covered advanced topics like memory allocation, pointers, and linked lists.	Substance and goal complexity	Revised course contents and moved the focus to basic programming concepts (see 2.1).	2005	2006
2.4	Students had problems to memorize the tasks that were done in the lectures, since programming involves many processes and requires a lot of detailed information.	Substance and implementation complexity	Recorded the lectures, the events on the lecturer laptop and talk, to offer support for different learning styles and allow checkup of details from the lectures later (see 1.4).	2005	2008
2.5	Students had problems understanding programs.	Substance and implementation complexity	Introduced a new program visualization tool [Rajala et al. 2009] with voluntary assignments; assignments became compulsory in 2009.	2008	2008 2009
2.6	The course used an HP-UNIX C-compiler, since the VLE used it for testing the assignments. In the labs, the students accessed UNIX workstations from Windows PCs with terminal connection software, even though students were not well versed in UNIX.	Technical and implementation complexity	Adopted a new integrated development environment with a simple editor, interpreter, and debugger that was available for most operating systems [Python Software Foundation 2010]. Students could use the same tools at home that were used in the labs, in lectures, and by the VLE.	2005	2006

Table V. Continued

ID	Problem Description	Problem Category	Actions Taken	Observed	Acted Upon
2.7	Students used C-compilers that they found on the Internet and different default compilation arguments led to different program behavior in the PCs and in the VLE.	Technical and implementation complexity	Started using an open source development environment that was freely available for all the parties involved in the course, also for the students (see 2.6), and fixed the tools used and their versions in the beginning of the course.	2005	2006
2.8	Course project focused on topics that were introduced late in the course and the project deadline was set a month after the course closing.	Implementation complexity	Introduced a new project focusing on concepts discussed earlier in the course and moved the the project deadline to the end of the course.	2005	2005
2.9	Students observed many defects and improvement needs in the assignments, the study guide, and the course in general.	Implementation complexity	Fixed defects as found, addressed improvement needs as feasible, and started rewarding students for reporting the defects.	2006	2006 2007
2.10	The main study material in the course was the lecture slides with links to websites and discrete books on the C programming language. This made it hard to give students clear study assignments and examples to look at for help with specific problems.	Implementation complexity	Developed a Finnish language Python programming guide [Kasurinen 2008] and made it freely available as a PDF-file; the guide became the definitive study material for the course focusing on the course contents only.	2005	2006
2.11	To assure that every student completed the assignments and project by her/himself, students had individual assignments, which limited collaboration also in learning.	Implementation complexity	Introduced an Internet Relay Chat channel for the course to allow discussions among the students and with the teaching assistants in a controlled environment.	2007	2007 2008
2.12	Course grades varied a lot and had a a distribution resembling a two-humped camel.	Implementation complexity	Experimented with a point collection system without fixed quotas for different activities.	2007	2009
3.1	Increasing the amount of student assignments and tracking of progress required a lot of effort both before and during each course.	Course maintenance effort	Elimination of the course variants started reducing the maintenance effort (see 2.1) and the course revision planned for 2010 (see 3.2) has the same aim.	2006	2007
3.2	Python 3 released and announced to be incompatible with version 2 which made a manual code review unavoidable.	Course maintenance effort	Planned technical infrastructure upgrade for the course of 2010.	2009	2010
3.3	The teaching assistant actively implementing improvements in the course left the project.	Course resource reduction	Reduced the improvement plans and focused on smaller issues.	2008	2008
3.4	The completion of the prerequisite courses was not checked when students enrolled in new courses.	Lack of systemic oversight	Tried to achieve better course results, passing the course with one enrollment, since could not introduce extrinsic motivators.	2005	2006 2008

were observed to include four kinds of complexities—goal, substance, implementation, and technical complexities—which all had elements of both essential and accidental complexity [cf. Brooks 1987]. This made course completion unnecessarily difficult for students.

Third, part of the student behavior can be explained by intrinsic motivation. Throughout the history of the course, some students had always completed it successfully, meaning that some students have been intrinsically motivated, or skilled enough, to satisfy the course requirements irrespective of the complex arrangements. The importance of intrinsic motivation was demonstrated by the increase in the number of completed projects in 2007, from 81% to 92%, and by the fact that 23 students completed the game-like Turtlet project even though they did not complete all the other compulsory assignments. While some students equally criticized the Turtlet project, the criticism can also be explained by intrinsic motivation and by examining its four aspects: challenge, curiosity, control, and context [Lepper and Henderlong 2000]. Lepper and Henderlong note that "Bruner [1961, 1966] wrote of the importance of the contextualization of learning—of students' being able to see, for example, the relevance and utility of the skills they are being taught in school for solving problems or accomplishing goals of their own, objectives they would find of inherent personal interest, in the larger world outside of their classrooms" [Lepper and Henderlong 2000]. That is, some would-be engineers in the course did not find the project useful from their future career or from their present study point of views. The project also limited their creativity and control over the project, by establishing a framework that dictated the program structure. Some students were familiar with the LOGO programming language [Logo Foundation 2010] and due to the visual similarities between the two, these students exhibited limited curiosity toward the project. Finally, some skilled programmers took the course each year, and they did not find the project sufficiently challenging. McWhorter and O'Connor [2009] report similar motivational experiences from using LEGO Mindstorms in CS1.

Another part of the student behavior relates to extrinsic motivation. The students that enrolled in the course due to extrinsic motivators, such as because it was a curriculum requirement, as opposed to enrolling due to intrinsic interest in the topic, were further impeded by the course complexity. Since the passing of the course was only checked before the students graduated and, in the absence of any systematic checking of the completion of the prerequisite at the time of the enrollment in the later courses, students could effectively decide for themselves when they actually completed the course.

Finally, many students dropped the course due to hygiene problems. As the course complexities started to accumulate, an increasing number of students dropped the course and blamed the accidental complexities: "the course does not have a book I could study," "you need to know UNIX to do the programming assignments," and "I cannot attend the lectures since they overlap with the lectures of another course." The fact that a 5-ECTS course represents only 8% of the expected annual workload (60 ECTS [European Commission 2009]) also meant that dropping one course a year was generally not considered very dramatic by the students.

Thus, the lack of intrinsic motivation, combined with the course failure to motivate the students about the course contents, created a situation in which the course focus moved from the contents to the course arrangements. Herzberg [1968] notes that dissatisfaction-avoidance, the hygiene factors, are extrinsic to the job itself and, in the CS1 course context, they can be defined as the accidental complexities in the course arrangements. In short, the students who were not motivated to complete the course focused on the problems in the course arrangements, got dissatisfied, dropped the course, and justified their decision using the hygiene problems.

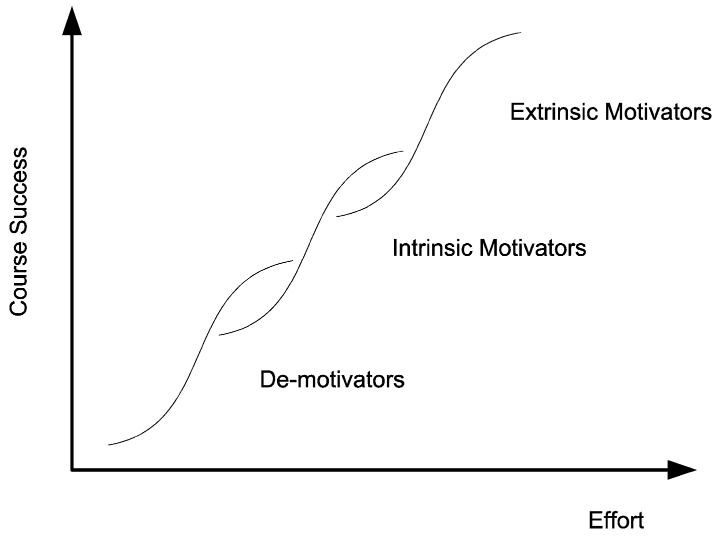


Fig. 9. The Three-Motivator Theory: course success can be improved by removing de-motivators, increasing intrinsic motivators or increasing extrinsic motivators, as appropriate in the course context.

5.1.2 How Can the Pass Rate Be Improved? In this study of a troubled course, the most useful way to improve the course results was to eliminate the hygiene problems, the de-motivators leading to dissatisfaction among the students. The main source of the hygiene problems, the complexities in the course arrangements, was tackled by the elimination of accidental and undue complexity, and the essential complexity was reduced where possible. Attempts were also made to improve the intrinsic and extrinsic motivation of the course. The intrinsic motivation was addressed by adopting programming tools and by revising the assignments and project in ways that would be considered more interesting and useful for the students. The extrinsic motivation was handled through the course policies, such as no longer accepting assignments from previous years, requiring students to complete at least 40% of the assignments on a weekly schedule, and requiring the submission of an initial version of the course project three weeks before the final submission. Despite the mixed reception of the intrinsic motivators, the extrinsic motivators increased the predictability of the student behavior among those students who had started to do the assignments.

The de-motivators, together with the intrinsic and extrinsic motivators, played important roles in the improvement of the pass rate for the course. In particular, the following actions to rehabilitate a troubled first programming course were observed to work well (listed in the order in which they were tackled).

(1) Eliminate de-motivators by resolving hygiene problems.
(2) Increase intrinsic motivators in the course by making it more interesting and useful.
(3) Introduce extrinsic motivators to increase the predictability of student behavior.

These three steps are visualized in Figure 9 as the Three-Motivator Theory. This theory is developed based on the analogous idea of multiple technology S-Curves contributing to overall system performance improvement [Foster 1986], where technology is defined in its broadest sense as the processes, tools, and practices used in the course. Each new technology aims to eliminate de-motivators, increase intrinsic motivators, or increase extrinsic motivators and should, consequently, improve the system performance, in this case the course pass rate. Improvement is slow at first as the technology is adjusted to the course context and, after a period of steady improvement, the performance improvement degrades as the technology approaches its natural limit. In the present study, no attempt was made to establish the limits of different technologies and motivators but, since the reasons for dropping the course varies for different students (Table III and Section 5.1.3), each motivator can only eliminate part of the cause for dropouts. Further, maximizing the pass rate requires addressing all three motivators. The study therefore demonstrates an alternative to the search for a silver bullet [Brooks 1987]: elimination of de-motivators, such as too few programming assignments and a lack of study materials, can be combined with the introduction of intrinsically interesting assignments and projects, and course policies can be used to increase the predictability of the student behavior. Since the form and number of (de-)motivators is only limited by human imagination, the Three-Motivator Theory should allow for the achievement of a desired pass rate. The technology S-curve based performance improvement has been previously demonstrated in industries like software development [Nikula et al. 2010], disk-drives [Christensen 1992], ocean-liners [Foster 1986], and artificial hearts [Foster 1986].

The order of focusing on the different motivators can vary between organizations and situations. However, de-motivators would appear to serve as a reasonable first step, since they focus on the problems that can actively decrease the course success. Intrinsic motivators are the natural reason for learning, but since not all learning is or can be made intrinsically motivating for everyone, "some use of extrinsic rewards may be inevitable" [Lepper and Henderlong 2000]. The motivator type to be tackled first should be the one that constrains the success the most at present.

5.1.3 Rival Explanations. The previous sections summarized the key factors influencing the low pass rate and how they were improved, but a case study cannot be considered complete before the reader is convinced that all the relevant information has been collected and considered [Yin 2003]. Thus, the absence of rival explanations in reporting the course dropouts is first justified and then the main rival explanations for the improvement are discussed below.

Numerous other papers have studied the reasons for dropouts and, for example, Kinnunen and Malmi's [2006] study on the first programming course reports reasons similar to those reported in the annual dropout surveys conducted in the present study: lack of time, lack of motivation, perceived difficulty of the course, general difficulties with time management and the planning of studies, and the decision to prefer something else. Xenos et al. [2002], on the other hand, studied a distance course on Informatics and grouped the dropout reasons into five major categories: professional (62%), academic (46%), family (18%), health (10%), and personal reasons (9%). Bennett [2003] equally reports that the student decisions to stay or quit in a large Business Studies Department were affected by reasons such as financial hardship, personal problems, commitment, satisfaction, the extent of personal investment, self-esteem, academic performance, motivation, and stress. Overall, it is evident that dropouts can be caused by problems in almost any aspect of a student's educational experience, or simply due to life circumstances, and an appropriately devised study is likely to surface them. Thus, rather than just report the reasons for dropouts so as to compare with other studies, the present study focused on the process of identifying the problems holistically in the course context and on how to rehabilitate the course.

A more controversial issue is the cause of the improved pass rate. Since the improvement took place gradually, rather than abruptly, it is assumed that no single factor can explain the improvement and the focus is on exploring alternative factors for the ones reported in Section 5.1.2. The main rival explanations for the improved results include the changes in the course personnel, course simplification, the procedural programming paradigm adopted, the programming language used, the Hawthorne effect [Landsberger 1968], and student characteristics.

All the teaching assistants for the course changed approximately every two years, so no single assistant was irreplaceable. The instructor, on the other hand, remained constant over the study period. Since we have not studied the effect of the instructor on the course outcome, we can only report our personal observations and do so by focusing on two events. First, while the course results were much better by the end of the study than before it, the performance was even worse than prior for the first course taught by the new instructor. This initial performance dip in 2005 could be seen as a sign of the new instructor's negative impact, but it could also be considered typical for any big change project (cf. the Classic Change Curve [Elrod and Tippett 2002; Nikula et al. 2010]). Second, the student feedback on the instructor varied from most negative to most positive comments over the course of the study period. However, to the best of our knowledge, courses with atypical arrangements tend to raise such extreme feelings as some students find the changes positive while others find them annoying. Overall, any good educator should be able to achieve similar results provided that he/she has the appropriate skills and resources (Section 5.2, the last paragraph) as well as a desire to go through a rehabilitation exercise such as the one reported in this study.

Second, the course simplification apparently affected the results but, by looking at the current course contents (Table IV) and by comparing this with typical introductory programming course contents [Schulte and Bennedsen 2006], the course does not appear more trivial than any other. Compared with the Schulte and Bennedsen [2006] results, the course covers eleven of the listed topics, eight topics are covered to some extent, and nine topics are not in the course scope; the course covers the six most relevant topics but excludes the seven most difficult topics.

Third, even though the object oriented paradigm is important, we chose the procedural paradigm to assure that the students understood the basic sequential operation of the computer—all commands are executed sequentially in a predetermined order in the traditional von Neumann architecture. The students are exposed to objects also, since Python implements strings, lists, and files as objects.

Fourth, changing the programming language from C to Python simplified both the programming environment and the language constructs that students needed to learn. For example, Python has automatic memory management and garbage collection, so strings can be concatenated with the +-operator without manual memory operations. Similarly, a list object can be manipulated with *append* and *remove* methods without explicit links or pointers. Even though the transition to a simpler language was supported by many other actions aimed at making the learning easier—for example, a language specific programming guide was tailored for the course and made available in the native language, free of charge, and even supported by lecture videos later on—still not all of the students learned to program effectively or passed the course. Thus, even though the effect of using a simpler programming language was not studied explicitly, our expectation is that any programming language can become a hygiene factor, hindering the learning of unmotivated students, if it is not accompanied by supporting materials and other motivating factors. This transition alone cannot explain the outcome of the study.

Fifth, the Hawthorne effect refers to the tendency of people to work harder and perform better when they are observed as a part of an experiment [Landsberger 1968]. However, most of the students changed annually in the study and the role of the study was not emphasized to the students, even though they were informed about the continued course improvement work. Therefore, the Hawthorne effect is unlikely to have played a significant role in the study.

Finally, the student characteristics may have affected the study results in two ways. First, the study practices of the first year students appeared immature. For example, some students noted that they did not want a bad grade and preferred to drop the course and complete it later to avoid this situation. As a first-term course for first-year students the study practices must be assumed immature for all their university-level studies, so the course should try to help the students in acquiring solid study practices. Second, the characteristics of the students enrolling in each of the five courses during the period of the study can be assumed to differ due to the changes in the overall student population. This issue was studied by collecting data about the students' majors (Figure 6) and their number of previous course enrollments (Figure 8), but no factor explaining the results has been identified so far (cf. Guzdial and Soloway [2002]). While all of these factors played some role in this holistic course improvement study, these factors were deemed of lesser importance than the ones reported in Section 5.1.2.

5.2 Lessons Learned

The research questions focused on the motivational findings of the study and the systems approach that was used, the Theory of Constraints (TOC), provided the measurement and improvement frameworks for the study. The TOC focus on the system throughput, the pass rate, was a good fit for studying a troubled course, since it made it easy to visualize the results both annually and over time. Other courses may benefit from using more metrics that enable the continued tracking of course improvement, as demonstrated with the more explorative studies on programming skills (Figure 7). Overall, a high dropout rate appears to derive from three primary reasons: (1) the course arrangements not being on a par with the goals; (2) low student motivation for the course; or (3) the course being used as a "gatekeeper" course to filter students (e.g., Gill and Jones [2010] and McKinney and Denton [2004]). In a typical course, one expects the throughput to stabilize to a value that reflects the fit between the course arrangements and the student motivation.

The TOC approach to considering the system as a sequence of steps was implemented in this study by categorizing the course deliverables into weekly assignments, quizzes, project, and exam. This division helped to identify the course project as a bottleneck, constraining the pass rate in 2006, and led to a new project in 2007 with an increased, albeit divided, interest. The longitudinal view on the student weekly workload (Figure 2) demonstrated the variance in the weekly workloads, leading to dissatisfaction among the students. Identifying the different causes of dissatisfaction is important since, even though such statistical fluctuation evens out for the course as a whole, the students that drop the course are lost for that year. For example, if 3% of the students encounter a problem each week that leads to them dropping the course, the course has only 65% of the original students remaining after fourteen weeks (i.e., 0.97^{14}).

The systemic course definition with course description, technical infrastructure, teaching, support, and assessment (Table II) served the study well. For example, it helped to manage all the changes when the technical infrastructure was revised (Section 4.3.2) and provided a clear starting point for the discussion on the course goals when the curriculum was revised (Section 4.3.3). The technical infrastructure included course assignments since their primary goal was to teach the programming concepts rather than to assess the skills. Also, policies were considered part of technical infrastructure since they provided the instructor with a tool to enforce different rules as a response to changes in the course and its environment (cf. Gill and Jones [2010]). The technical infrastructure was important in the study for three reasons. First, the course had 141 to 198 active students each year, so automation was required to manage and assess the students' weekly assignments. Second, the course was an introductory course with most of the students having no prior experience of programming or university study, which led to a need for definitive study materials to simplify the course. Third, the course was a practical course on programming, so the students had to be exposed to realistic tools. In general, the systemic course definition provided a holistic view of the course, one through which the course could be managed systematically and the viability of the proposed changes estimated and compared.

Finally, the study revealed the range of skills required of an instructor to take up holistic course rehabilitation. The study started from the instructor's vision that the course results could be substantially improved by changes that helped the students to learn better. The changes started with the discovery of a programming language that provided an improved fit for the course needs, but also required intrinsic and extrinsic motivators, an understanding of the hygiene factors, familiarity with the curriculum and related courses, an ability to implement changes in an organization prone to change resistance, and an idea of how to integrate multiple technologies in a holistic way. The initial course concept received limited support from both the departmental staff and the student body, mainly due to the adoption of a language that was not in mainstream industry use. Thus, the study demonstrates the importance of the course instructor having full responsibility for a course, and taking into account the needs of the different stakeholders from faculty, industry, and students, as well as the developments in the problem domain. An instructor rehabilitating a course needs the desire, the authority, and the resources to implement the changes successfully.

5.3 Implications of the Study

The study has implications at the course, department, and university levels. The study shows that the fit of a course to students can decrease over time due to developments in the environment, such as technology changes that change the expectations and required skills of the students. Thus, courses should be kept aligned with changes in the environment to keep them motivating for the students and relevant for other stakeholder groups. While eliminating the de-motivators that cause dissatisfaction among students can rehabilitate a troubled course, there is also a concomitant need to present the topics in an intrinsically motivating way. At the curriculum level, departments should develop explicit course sequences that build upon the skills acquired in other courses to increase the student intrinsic motivation to complete the courses as planned. The departments should also track the performance of their courses to be able to act on problems early, since rehabilitating a course is a lengthy process. At the university level, the study shows how the lack of systemic oversight at the highest level can undermine the lower level efforts to change a system. In particular, the university-level decision to waive the course prerequisite check in the study registry moved the operative decision of the course sequence to the students, which demonstrates the need to enforce compliance with extrinsic motivators if they are adopted. In general, a competent instructor may be able to deliver a successful course without all the proposed systemic frameworks. However, as the university teaching resources are cut and the interests of teachers diversify, such frameworks can prove invaluable to assure high quality and consistent teaching.

On the theoretical side, the study implications focus on the systems approach adopted. Even though the Theory of Constraints, with its measurement of system throughput, the pass rate, and the developed systemic course definition fit the study, four important questions remain. First, for a course with an acceptable pass rate, other metrics are needed to assess the course quality and the achievement of the learning objectives. Second, the focus of the present study on technical infrastructure in a troubled course raises a question as to whether the absence of de-motivators is a prerequisite for efficient consideration of the pedagogical aspects, like the learning styles used in a course. Third, the developed systemic course definition served well in an introductory technical course with a large student body, but its suitability for different kinds of course settings needs examination. Fourth, the inductively developed theory on rehabilitating a troubled course by eliminating de-motivators and increasing intrinsic as well as extrinsic motivators needs further empirical validation.

5.4 Limitations of the Study

The study has four high-level limitations. What started out as a technical study ended up as a motivational study, the study did not close with definitive cause-effect correlations, the course staff conducted the study, and the study focused only on the first programming course. Since the study started from the technical point of view, but turned to motivation, it demonstrates the importance of motivation in the educational setting. However, it does not cover the literature and data collection on motivation in the depth that one would expect in a pure study of motivation. The study explored cause-effect relationships but did not result in a comprehensive framework [cf. Golding et al. 2009] to identify the correlations between all the factors. The number of factors observed was large and their selection process included many case specific factors. Gill and Jones [2010] warn about conducting statistical analyses with insufficient understanding of the problem, and emphasize the importance of in-depth case studies on how to improve the course success. Yin [2003] also supports this view by stating that case studies are typically used "to *explain* the presumed causal links in real-life interventions that are too complex for the survey or experimental strategies." Third, the researcher bias is an evident risk in a study conducted by a participant-observer [Yin 2003]. It is clear that, for example, a lecturer cannot observe the lectures objectively, and "may, at times, have to assume certain positions or advocacy roles contrary to the interests of good scientific practice" [Yin 2003]. However, the access to all the student assignments, exams, grades, survey results, as well as direct feedback from the students, provides for such a rich set of data that it makes it possible to conduct an objective study in which the course staff do not need to play a central role. Finally, the study has not explored the student programming skills on subsequent courses, even though anecdotal evidence from later course instructors was positive.

The data collection and analyses have four limitations. First, the course and the data collected have evolved during the study and direct comparison between the five case studies can be questioned. For example, moving from two course variants with two audiences to one course with one audience, but treating the results of the course variants en bloc is questionable. However, as the new course started to live a life of its own rather than be a descendant of the previous two courses, we suggest that the treatment is justified. Second, the final grade, as a derivative measurement, did not directly reflect the achievement of learning outcomes. The grading system therefore deserves a study of its own. Third, 35 exams and 25 surveys were administered in the course of the study, and the weekly programming assignments with example programs comprise a 30-page text document. The instruments used to collect data in the study are too numerous to publish in their entirety, but they are available for interested readers by contacting the first author of the paper. Fourth, the study does not provide definitive explanations of all the observed anomalies, like the drop in the enrollment count in 2005 and 2007, or the improved learning results in 2009. Based on the gathered evidence, some of these events were caused, to a certain extent, by contingent events in the wider context. Their further study would provide information of decreasing relevance for answering the primary research questions of this study.

6. CONCLUSION

This study set out to rehabilitate a troubled first programming course based on the Theory of Constraints and by using the pass rate as the success criterion. When the study started in 2005, a study period coinciding with when the course was taken over by the first author of this article, there were a priori problems with the pass rate and in the attainment of the learning outcomes, as well as a general atmosphere of dissatisfaction among the students. The pass rate had been 44% in 2004 but, after five years of study and course improvements, the pass rate reached 68% in 2009, a 55% improvement. During the same period, the course atmosphere had turned positive and the course received a student union excellence award in 2008 for its course materials. Based on these results, and the fact that introductory programming courses at the university level in general have an average pass rate of 67% [Bennedsen and Caspersen 2007], the course in question can be considered rehabilitated.

Since the course was run once a year from 2005 to 2009, each course was considered a single case study with replication logic addressing the most important problems observed in the previous year. The first course implementation in 2005 focused on problem analysis. It was followed by a major technical revision in 2006 and by curriculum level course changes in 2007. After these major changes, smaller changes were implemented in 2008 and 2009 to stabilize the course. The changes were managed and misfits were identified within a systemic course definition comprising the course description, technical infrastructure, teaching, support, and assessment elements. As a result of the study, three key reasons for the low pass rate were identified: programming as a discipline, course arrangements, and student behavior. The discipline of programming includes a certain amount of essential complexity, thereby requiring conscious effort to learn to program. The course arrangements were observed to include both essential and accidental complexities in the goals, substance, implementation, and technical solutions. Finally, the students were found to have motivational problems. The improvement in the course pass rate was achieved by tackling both the course and motivational problems, alongside changes in the course arrangements, according to the following three steps.

(1) Eliminate de-motivators by resolving hygiene problems.
(2) Increase intrinsic motivators in the course by making it more interesting and useful.
(3) Introduce extrinsic motivators to increase the predictability of student behavior.

The implemented changes focused on the technical aspects in the course arrangements that affected the student motivation to complete the course. However, it appears that to be the most effective, the extrinsic motivators need to be established at the organizational level and with supporting control systems rather than at the individual course level. Overall, the study demonstrates the importance of the holistic understanding of the system under study including its technical, human, and organizational aspects, as well as their dynamics to be able to induce a change within it.

Lee and Baskerville [2003] conclude that “the generalizability of empirical descriptions to theory is well developed” and, thus, the findings of the conducted case studies are generalized as the Three-Motivator Theory: course success can be improved by removing de-motivators, increasing intrinsic motivators or increasing extrinsic motivators, or any combination of these three, as appropriate in the course context. Since the study builds on the Two-Factor Theory developed to understand the human motivation to work, the new theory could also fit into other motivation guided contexts, such as organizational change involving learning. However, Lee and Baskerville [2003] also note that "The only way in which a researcher (or practitioner) may properly claim that the theory is indeed generalizable to the new setting would be for the theory to be actually tested and confirmed in the new setting."

In the course of 2009, the results started to approach the original improvement goal of an 80% pass rate. That is, the course pass rate was actually 80% if the reference point was considered to be only those students who submitted at least one weekly assignment, rather than to also include all those students who registered for the course but did not complete a single programming assignment for it. However, there is still a lot of room for improvement. Based on the Three Motivator Theory, it should be possible to achieve an 80% pass rate with all the students by eliminating more demotivators and improving intrinsic motivators, since we have improvement ideas that have not yet been put into practice. It further appears reasonable to aim at a 90% pass rate by introducing additional extrinsic motivators but, since dropouts also result from changes in family, health and personal situations, there are limitations to what motivators can ultimately achieve. Still, we find the goal of an extraordinarily motivating first programming course worthwhile, since it holds the potential to attract new students to the field of computing and to convey the broader value of programmatic thinking to their wider studies, future workplace careers, and everyday life.

ACKNOWLEDGMENTS

We would like to thank all the students taking this course over the five-year study period, not only for their forbearance in completing all the surveys, but also for tolerating the ups and downs of change. We would also like to express our gratitude to the IT Department at LUT for all the resources provided to conduct the study and the following people for their comments on an earlier version of this article: Donald C Gause, Jorma Sajaniemi, Heikki Kälviäinen, Kari Smolander, Jari Porras, Arto Kaarna, Tero Pesonen, and Erno Vanhala. Finally, we would like to thank the anonymous reviewers and the editors of TOCE for providing comments on an earlier version of this article, which have helped to improve the article substantially.

REFERENCES

ALEXANDER, C. 1964. *Notes on the Synthesis of Form*. Harvard University Press.

AMBROSE, M. L. AND KULIK, C. T. 1999. Old friends, new faces: Motivation research in the 1990s. *J. Man. 25*, 3, 231–292.

ANDERSON, D. J. 2004. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall PTR, Upper Saddle River, New Jersey.

BAKER, R. S. J., CORBETT, A. T., AND ALEVEN, V. 2008. More accurate student modeling through contextual estimation of slip and guess probabilities in Bayesian knowledge tracing. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS'08)*. 406–415.

BECKER, K. 2002. Back to Pascal: Retro but not backwards. *J. Comput. Sci. Coll. 18*, 2, 17–27.

BENNEDSEN, J. AND CASPERSEN, M. E. 2007. Failure rates in introductory programming. *SIGCSE Bull. 39*, 2, 32–36.

BENNETT, R. 2003. Determinants of undergraduate student drop out rates in a university business studies department. *J. Further High. Educ. 27*, 2, 123.

BERGIN, S. AND REILLY, R. 2005. The influence of motivation and comfort-level on learning to program. In *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group (WPPI'05)*. 293–304.

BILLS, D. P. AND CANOSA, R. L. 2007. Sharing introductory programming curriculum across disciplines. In *Proceedings of the 8th ACM SIGITE Conference on Information Technology Education (ITE'07)*. 99–106.

BOISVERT, C. R. 2009. A visualisation tool for the programming process. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'09)*. 328–332.

BONWELL, C. C. AND EISON, J. A. 1991. *Active Learning: Creating Excitement in the Classroom*. The George Washington University, School of Education and Human Development, Washington D.C.

BÖSZÖRMÉNYI, L. 1998. Why Java is not my favorite first-course language. *Softw. - Concepts Tools 19*, 3, 141–145.

BOYD, L., GUPTA, M., AND SUSSMAN, L. 2001. A new approach to strategy formulation: Opening the black box. *J. Educ. Bus. 76*, 6, 338–344.

BROOKS JR., F. P. 1987. No silver bullet - Essence and accidents of software engineering. *Comput. 20*, 4, 10–19.

BRUCE, K. B. 2005. Controversy on how to teach CS 1: A discussion on the SIGCSE-members mailing list. *SIGCSE Bull. 37*, 2, 111–117.

BRUNER, J. S. 1961. The act of discovery. *Harvard Educ. Rev. 31*, 21–32.

BRUNER, J. S. 1966. *Toward a Theory of Instruction*. Harvard University Press.

CASPERSEN, M. E. AND KOLLING, M. 2009. STREAM: A first programming process. *Trans. Comput. Educ. 9*, 1, 1–29.

CHRISTENSEN, C. M. 1992. Exploring the limits of the technology S-Curve. Part I: Component technologies. *Prod. Oper. Man. 1*, 4, 334–357.

COOPER, M. J. AND LOE, T. W. 2000. Using the theory of constraints' thinking processes to improve problem-solving skills in marketing. *J. Market. Educ. 22*, 2, 137–146.

DAVIS, A. M. 2003. The art of requirements triage. *Comput. 36*, 3, 42–49.

DEMING, W. E. 1990. *Out of the Crisis*. Massachusetts Institute of Technology, Cambridge, MA.

DESHIELDS, O. W. J., KARA, A., AND KAYNAK, E. 2005. Determinants of business student satisfaction and retention in higher education: Applying Herzberg's two-factor theory. *Int. J. Educ. Man. 19*, 2, 128–139.

DETTMER, H. W. 1997. *Goldratt's Theory of Constraints: A Systems Approach to Continuous Improvement*. ASQ Quality Press, Wilwaykee, WI.

DOUCE, C., LIVINGSTONE, D., AND ORWELL, J. 2005. Automatic test-based assessment of programming: A review. *J. Educ. Res. Comput. 5*, 3, 13.

DOUGLAS, J., MCCLELLAND, R., AND DAVIES, J. 2008. The development of a conceptual model of student satisfaction with their experience in higher education. *Qual. Assur. Educ. 16*, 1, 19–35.

EASTON, G. 2010. Critical realism in case study research. *Indust. Market. Man. 39*, 1, 118–128.

EBRAHIMI, A. 1994. Novice programmer errors: Language constructs and plan composition. *Int. J. Hum.-Comput. Stud. 41*, 4, 457–480.

EISENHARDT, K. M. 1989. Building theories from case study research. *Acad. Man. Rev. 14*, 4, 532–550.

ELROD, P. D. I. AND TIPPETT, D. D. 2002. The "Death Valley" of change. *J. Org. Change Man. 15*, 3, 273–291.

EUROPEAN COMMISSION. 2009. ECTS users' guide. Report, Office for Official Publications of the European Communities, Luxembourg, Belgium.

EUROPEAN COMMISSION. 2010. The Bologna process - Towards the European higher education area. http://ec.europa.eu/education/higher-education/doc1290_en.htm.

FORTE, A. AND GUZDIAL, M. 2005. Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Trans. Educ. 48*, 2, 248–253.

FOSTER, R. N. 1986. *Innovation: The Attacker's Advantage*. Summit Books, New York.

FURNHAM, A., ERACLEOUS, A., AND CHAMORRO-PREMUZIC, T. 2009. Personality, motivation and job satisfaction: Hertzberg meets the Big Five. *J. Man. Psych. 24*, 8, 765–779.

FURUGORI, T. AND JALICS, P. 1977. First course in computer science, a small survey. In *Proceedings of the 7th SIGCSE Technical Symposium on Computer Science Education (SIGSCE'77)*. 119–122.

GILL, T. G. AND JONES, J. 2010. A tale of three classes: Case studies in course complexity. *J. Inf. Technol. Educ. 9*, 29.

GLASER, B. AND STRAUSS, A. L. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago.

GOLDING, P., DONALDSON, O., AND TENNANT, V. 2009. Application of modified perceived learning problem inventory (PLPI) to investigate performance in introductory programming. In *Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference (FEC'09)*. 366–371.

GOLDRATT, E. M. 1994. *What Is This Thing Called Theory of Constraints and How Should It Be Implemented?* North River Press, Great Barrington, MA.

GOLDRATT, E. M. AND COX, J. 2004. *The Goal: A Process of Ongoing Improvement* 3rd Ed. North River Press, Great Barrington, MA.

GOLDRATT, R. AND WEISS, N. 2005. Significant enhancement of academic achievement through application of the Theory of Constraints (TOC). *Hum. Syst. Man. 24*, 1, 13–19.

GROLLMAN, W. K. 1974. Hygiene factors in professional education programs. *J. Account. 137*, 1, 85–88.

GUZDIAL, M. 2009. Education: Teaching computing to everyone. *Comm. ACM 52*, 531–33.

GUZDIAL, M. AND SOLOWAY, E. 2002. Teaching the Nintendo generation to program. *Comm. ACM 45*, 4, 17–21.

HACKMAN, J. R. AND OLDHAM, G. R. 1976. Motivation through the design of work: Test of a theory. *Org. Behav. Hum. Perform. 16*, 2, 250–279.

HALL, T., BADDOO, N., BEECHAM, S., ROBINSON, H., AND SHARP, H. 2009. A systematic review of theory use in studies investigating the motivations of software engineers. *Trans. Softw. Eng. Methodol. 18*, 3, 1–29.

HERZBERG, F. 1968. One more time: How do you motivate employees? *Harvard Bus. Rev. 46*, 1, 53–62.

HERZBERG, F., MAUSNER, B., AND SNYDERMAN, B. B. 1959. *The Motivation to Work*. Wiley, New York.

HIGGINS, C. A., GRAY, G., SYMEONIDIS, P., AND TSINTSIFAS, A. 2005. Automated assessment and experiences of teaching programming. *J. Educ. Res. Comput. 5*, 3, 21.

HSU, P.-F. AND SUN, M.-H. 2005. Using the Theory of Constraints to improve the identification and solution of managerial problems. *Int. J. Man. 22*, 3, 415–425.

HUME, G., MICHAEL, J., ROVICK, A., AND EVENS, M. 1996. Hinting as a tactic in one-on-one tutoring. *J. Learn. Sci. 5*, 1, 23–47.

JENKINS, T. 2001. The motivation of students of programming. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'01)*. 53–56.

JENKINS, T. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Learning and Teaching Support Network Conference (LTSN-ICS'02)*. 53–58.

JIAU, H. C., CHEN, J. C., AND SSU, K.-F. 2009. Enhancing self-motivation in learning programming using game-based simulation and metrics. *IEEE Trans. Educ. 52*, 4, 555–562.

JOHNSON, L. F. 1995. C in the first course considered harmful. *Comm. ACM 38*, 5, 99–101.

JOINT TASK FORCE FOR COMPUTING CURRICULA. 2001. Computing Curricula 2001 for Computer Science. http://www.acm.org/education/curricula.html.

JOINT TASK FORCE FOR COMPUTING CURRICULA. 2004. Software Engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering. http://www.acm.org/education/curricula.html.

KASURINEN, J. 2006. *Python as a Programming Language for the Introductory Programming Courses*. Bachelor thesis (In Finnish). Department of Information Technology, Lappeenranta University of Technology, Finland.

KASURINEN, J. 2008. *Python Programming Guide version 1.2* (In Finnish). Report 12. Lappeenrannan University of Technology, Finland.

KASURINEN, J. AND NIKULA, U. 2007. Lower dropout rates and better grades through revised course infrastructure. In *Proceedings of the 10th International Conference on Computers and Advanced Technology in Education* (IASTED'07). 152–157.

KASURINEN, J. AND NIKULA, U. 2009. Estimating programming knowledge with Bayesian knowledge tracing. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'09)*. 313–317.

KASURINEN, J., PURMONEN, M., AND NIKULA, U. 2008. A study of visualization in introductory programming. In *Proceedings of the Annual Meeting of the Psychology of Programming Interest Group (PPIG'08)*. 181–194.

KAUFFMAN, S. 1993. *Origins of Order: Self Organization and Selection in Evolution*. Oxford University Press.

KELLEHER, C. AND PAUSCH, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *Comput. Surv. 37*, 2, 83–137.

KINNUNEN, P. AND MALMI, L. 2006. Why students drop out CS1 course? In *Proceedings of the 2nd International Workshop on Computing Education Research (CER'06)*. 97–108.

KINNUNEN, P. AND MALMI, L. 2008. CS minors in a CS1 course. In *Proceedings of the 4th International Workshop on Computing Education Research (CER'08)*. 79–90.

KINNUNEN, P., MCCARTNEY, R., MURPHY, L., AND THOMAS, L. 2007. Through the eyes of instructors: A phenomenographic investigation of student success. In *Proceedings of the 3rd International Workshop on Computing Education Research (CER'07)*. 61–72.

LAHTINEN, E., ALA-MUTKA, K., AND JÄRVINEN, H.-M. 2005. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'05)*. 14–18.

LANDSBERGER, H. A. 1968. *Hawthorne Revisited: Management and the Worker, Its Critics, and Developments in Human Relations in Industry*. Cornell University, Ithaca, NY.

LEE, A. S. AND BASKERVILLE, R. L. 2003. Generalizing generalizability in information systems research. *Inf. Syst. Res. 14*, 3, 221–243.

LEPPER, M. R. AND HENDERLONG, J. 2000. Turning "Play" into "Work" and "Work" into "Play": 25 years of research on intrinsic versus extrinsic motivation. In *Intrinsic Motivation: Controversies and New Directions*, C. Sansone and J. Harackiewicz Eds., Academic Press, San Diego, 257–307.

LLOYD III, J. T. AND LANA, C. 2003. Goldratt's thinking process applied to the budget constraints of a Texas MHMR facility. *J. Health Hum. Serv. Admin. 26*, 3/4, 416–437.

LOGO FOUNDATION. 2010. Logo Foundation. Available online at http://el.media.mit.edu/logo-foundation/.

MABIN, V. J. AND BALDERSTONE, S. J. 2003. The performance of the theory of constraints methodology: Analysis and discussion of successful TOC applications. *Int. J. Operat. Prod. Man. 23*, 5/6, 568–595.

MASLOW, A. 1954. *Motivation and Personality*. Harper and Row, New York.

MCIVER, L. AND CONWAY, D. 1996. Seven deadly sins of introductory programming language design. In *Proceedings of the International Conference on Software Engineering: Education and Practice (CSE'96)*. 309–316.

MCKINNEY, D. AND DENTON, L. F. 2004. Houston, we have a problem: There's a leak in the CS1 affective oxygen tank. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'04)*. 236–239.

MCWHORTER, W. I. AND O'CONNOR, B. C. 2009. Do LEGO Mindstorms motivate students in CS1? In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE'09)*. 438–442.

MILLER, G. A. 1956. The magical number 7 plus or minus two: Some limits on our capacity for processing information. *Psych. Rev. 63*, 2, 81–97.

MOW, I. T. C. 2008. Issues and difficulties in teaching novice computer programming. In *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, M. Iskander Ed., Springer, 199–204.

MYLLER, N., BEDNARIK, R., SUTINEN, E., AND BEN-ARI, M. 2009. Extending the engagement taxonomy: Software visualization and collaborative learning. *Trans. Comput. Educ. 9*, 1, 1–27.

NIKULA, U., JURVANEN, C., GOTEL, O., AND GAUSE, D. C. 2010. Empirical validation of the Classic Change Curve on a software technology change project. *Inf. Softw. Technol. 52*, 6, 680–696.

PIRINEN, T. 2008. *The Study Record Analysis of Fundamentals of Programming Course*. Bachelor thesis (In Finnish). Department of Information Technology, Lappeenranta University of Technology, Finland.

PLAZA, I. AND MEDRANO, C. T. 2007. Continuous improvement in electronic engineering education. *IEEE Trans. Educ. 50*, 3, 259–265.

POLITO, T., WATSON, K., AND VOKURKA, R. J. 2006. Using the Theory of Constraints to improve competitiveness: An airline case study. *Compet. Rev. 16*, 1, 44–50.

PYTHON SOFTWARE FOUNDATION. 2010. Python Programming Language - Official Website. http://www.python.org/.

RAADT, M. D., WATSON, R., AND TOLEMAN, M. 2003. Language tug-of-war: Industry demand and academic choice. In *Proceedings of the 5th Australasian Conference on Computing Education - Volume 20 (ACE'03)*. 137–142.

RAJALA, T., LAAKSO, M.-J., AND KAILA, E. 2009. ViLLE - The Visual Learning Tool. http://ville.cs.utu.fi/.

REHMAN, H.-U., SAID, R. A. A., AND AL-ASSAF, Y. 2009. An integrated approach for strategic development of engineering curricula: Focus on students' design skills. *IEEE Trans. Educ. 52*, 4, 470–481.

ROBERTS, E. S. 1993. Using C in CS1: evaluating the Stanford experience. In *Proceedings of the 24th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'93)*. 117–121.

RODRIGO, M. T., MA, AND BAKER, R. S. J. D. 2009. Coarse-grained detection of student frustration in an introductory programming course. In *Proceedings of the 5th International Workshop on Computing Education Research Workshop (CER'09)*. 75–80.

SCHULTE, C. AND BENNEDSEN, J. 2006. What do teachers teach in introductory programming? In *Proceedings of the 2nd International Workshop on Computing Education Research Workshop (CER'06)*. 17–28.

SIRIAS, D. 2002a. Using graphic organizers to improve the teaching of business statistics. *J. Educ. Bus. 78*, 1, 33–37.

SIRIAS, D. 2002b. Writing MIS mini-cases to enhance cooperative learning: A Theory of Constraints approach. *J. Inf. Sys. Educ. 13*, 4, 351–356.

SLEEMAN, D. 1986. The challenges of teaching computer programming. *Comm. ACM 29*, 9, 840–841.

SOH, L.-K., SAMAL, A., AND NUGENT, G. 2005. A framework for CS1 closed laboratories. *J. Educ. Res. Comput. 5*, 4, 2.

STRAUSS, A. L. AND CORBIN, J. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* 2nd Ed. Sage Publications, Thousand Oaks, CA.

THWEATT, M. 1994. CSI closed lab vs. open lab experiment. In *Proceedings of the 25th SIGCSE Symposium on Computer Science Education (SIGCSE'94)*. 80–82.

UMBLE, M. AND UMBLE, E. 2000. Manage your projects for success: An application of the Theory of Constraints. *Prod. Inven. Man. J. 41*, 2, 27–32.

WALKER II, E. D. AND COX III, J. F. 2006. Addressing ill-structured problems using Goldratt's thinking processes: A white collar example. *Man. Decis. 44*, 1, 137–154.

WIEDENBECK, S., LABELLE, D., AND KAIN, V. N. R. 2004. Factors affecting course outcomes in introductory programming. In *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG'04)*. 97–110.

WILSON, B. C. AND SHROCK, S. 2001. Contributing to success in an introductory computer science course: A study of twelve factors. *SIGCSE Bull. 33*, 1, 184–188.

VIOPE SOLUTIONS LTD. 2010. Viope Solutions. http://www.viope.com.

VROOM, V. H. 1964. *Work and Motivation*. Wiley, New York.

XENOS, M., PIERRAKEAS, C., AND PINTELAS, P. 2002. A survey on student dropout rates and dropout causes concerning the students in the Course of Informatics of the Hellenic Open University. *Comput. Educ. 39*, 4, 361–377.

YIN, R. K. 2003. *Case Study Research: Design and Methods* 3rd Ed. Sage Publications, Thousand Oaks, CA.

Received October 2010; revised July 2011; accepted August 2011