**SDPS-2021**
Printed in US, December 2021
©2021 Society for Design and Process Science

# TEACHING SOFTWARE DEVELOPMENT FOR PERVASIVE COMPUTING ENVIRONMENT

**Joakim Bjørk and Radmila Juric**
**University of South Eastern Norway**
**Department of Science and Industry systems**
**Faculty of Technology, Natural Sciences, and Maritime Sciences{Joakim.Bjork@usn.no; Radmila.Juric@usn.no}**

## ABSTRACT

There are many attempts in academic curriculum which address rapid and constant changes in computer science, but they are mostly based on prescriptive advice and processes known in the management of regulated education. This paper looks at the problem of introducing novel subjects in the existing curriculum of computer engineering, with one important goal in mind: could we deliver the latest practices in software development, based on the advanced software technologies and applications, within relatively rigid and static academic program which dictates the structure and content of the curriculum. The paper also asks questions on teaching and-learning practices in such situations and questions our redlines to break with traditional tuition in conventional classroom, and, at the same rise awareness that teachers may become redundant in the process of learning. Finally, while sitting comfortable in our student-centered learning environment of the 21st century, we ask which exact tuition would modern students need to use their computational literacy in resolving problems in computer science and engineering and when does the learning happen? The example described in this paper if taken from the USN Department of Science and Industry Systems.

## INTRODUCTION

Computer science education is at its crossroads. In the last couple of decades, the rapid advances in software and communication technologies, and the adoption of mobile and wireless computing, as the backbone of modern computational power, changed our lives beyond recognition. However, it has also put pressure on educators, responsible for creating curricula. The urgent integration of computer science at any level of education is a sine qua non and computational thinking (Saidin et al., 2021), (Lockwood and Mooney, 2017) has been considered as the most basic skill in this century.

The motivation for writing this paper is threefold.

- Sharing experiences and visions on incorporating some aspects of pervasive computing into our curriculum which has traditional academic approach in delivering software engineering program.
- Examining the (Tissenbaum and Ottenbreit Leftwhich, 2020) publication in which the vision of computer science education for 2030 is outlined and would like to see if we can take the first steps in regulated education systems of Western Europe.
- Addressing messages from the industry where voices on "Becoming and Adaptive Experts" (Burke and Balley, 2020) are very loud and would like to see how we can address the same views in education. We would like to see if we can address the issue that industry needs in terms of "*creating graduates which can combine diverse specialization rather than having a routine knowledge in a special academic domain*".

These three bullets above put enormous pressure on any regulated education and might trigger rethinking on how to create academic programs, manage constant changes in technologies and take into account demands from industry.

The paper is organized as follows. In the next section we outline our undergraduate BSc Program in Software Engineering, give the program aims/goals and subjects we deliver. In the section which follows we talk about one subject on Software Modelling and Architectures which addresses the issues of developing Pervasive Computing Environment within the Curriculum in the light of the bullets above. The results are discussed, and we finish with conclusions.

## OULTINE OF OUR SOFTWARE ENGENEERING PROGRAM

The program is a 180 ECTS Bachelor program regulated by the Norwegian National Curriculum Regulations for Engineering Education. The focus of the program is on computer engineering and includes (amongst others) subjects on software development for cyber physical systems, and as such includes subjects on programming, software architecture, networks, operating systems and FPGA-programming in addition to mathematics and physics. Compulsory inclusions of mathematics and physics in the curriculum is dictated by the engineering profile of the program and is non-negotiable. Most of the subjects are delivered though lectures, laboratory work/exercises and workshops, and most of them include a formal / written exam. There are just a few subjects which include project-based assessment and their presentation, instead of the formal written exam.

A complete overview of the compulsory courses is given in Table 1. The 5th semester consists of elective subjects and students are encouraged to spend this semester abroad. Table 1 also shows a few interesting concepts of this program:

a) There are 10 subjects valued as 5 credits, which is 50% of all compulsory subjects;

b) There are not more than 3 subjects which are related to traditional computer science;
c) There are specialists' subjects which touch operating systems and networking (with a glimpse of "security) and could be seen as important in software engineering and
d) There is only subject which touches software development.

## THE PROBLEM

The subject on "Software Architectures and Modelling" has a self-explanatory title because the students are supposed to learn basic principles of software modeling and creating software architectures. However, whichever indicative syllabus and appropriate assessment we anticipate would work for the subject, there are numerous issues which surround them.

**Table 1 Compulsory Subjects**

| Name | Semester | ECTS | Assessment |
|---|---|---|---|
| The Engineering Role | 1 | 5 | Presentation |
| Computational programming | 1 | 5 | Written |
| Introduction to Linux | 1 | 5 | Written |
| Programming and Microcontrollers | 1 | 10 | Written |
| Digital Fundamentals | 1 | 5 | Written |
| Mathematics 1 | 2 | 10 | Written |
| Physics 1 – Mechanics | 2 | 5 | Written |
| Databases | 2 | 5 | Written |
| Object Oriented Programming | 2 | 10 | Written |
| Mathematics 2 | 3 | 10 | Written |
| Physics 2 – Electricity | 3 | 5 | Written |
| Algorithms and Data Structures | 3 | 5 | Written |
| Software Architectures and Modelling | 3 | 10 | Written |
| Statistics | 4 | 5 | Written |
| Operating Systems | 4 | 5 | Written |
| Systems Design and Engineering | 4 | 10 | Written & Presentation |
| Networks and Security | 4 | 10 | Written |
| Digital Circuits Synthesis | 6 | 10 | Written |
| Bachelor Thesis | 6 | 20 | Report and Presentations |

First, software modelling is impacted by numerous factors which range from

- the lack of standardized methodologies for software development,
- the abundance of software technologies which allow software deployment using a range of computational frameworks sitting on clouds, fogs, cloudlets, edges, and dust, and
- an unprecedent amount of data generated as we live and our expectations that we will always have computing programs which can process the data in any situation and at any time, to
- the dominance of pervasiveness in computing where boundaries between cyber artefacts and physical items are blurred and computationally powered devices are interwoven in our everyday lives.

The bullets above are our reality and they affect software modelling. Considering that we have only a software modelling language which was standardized in 2004 (OMG, 2004) and considering that software technologies drastically changed since 2004, then it is reasonable to expect that we must experience problems when teaching software modelling, if we wish to teach principles and practices of software development in 2021.

Second, creating Software Architectures (SA) (Bass et al., 2021) is another issue, but it does not bring forward the same problems as we outlined in the bullets above. The problem with SA is that it is an established discipline in computer science, defined in the late 90s and developed into a complex way of looking at constituent parts of software solutions across many problem domains. Apart from specific software architectural styles defined in the literature on SA, there are numerous issues which require examination of its efficiency, effectiveness, transparency, and various methods of its deployability using available technologies. In summary learning SA usually takes over any academic curriculum and requires to be delivered as a specialist program, possibly at the master's level.

In the light of the above, it is difficult to make a sound decision on (i) how exactly approach the delivery of these types of subjects and (ii) what we can expect students to learn. Without teaching software modelling principles and highlighting the role SA have in them, we could not claim that we are delivering an academic curriculum which covers all the aspects of computer engineering. Also, by avoiding the issue of pervasiveness and not talking about the modern aspects of computations we face in everyday life, we will deprive student from understanding the new role the data and computing algorithms have in pervasive spaces. Students will not be ready for facing problems of "everyday computational principles" when they leave the University and start building their professional careers. If we add to this our infatuation with predictive and learning technologies and algorithms which shape the current definition of Artificial Intelligence (AI), then we can clearly see the scale of the current teaching problems.

Finally, it has been known for a decade that we can not teach modern software development principles by having a handy published book as either a textbook or a book which can be used as a support in teaching and learning. Books published since mid-90s and through 2000, when the modeling language UML was standardized, are dangerous to use in 2021. It is not that they have incorrect presentation of UML syntax and semantics. It is the examples which books offer for the purpose of teaching software abstractions using UML modeling concepts. They are dangerously out of date. Some examples in these books are non-existent in real life. If we add to this the problem that we have no standardized methodology for software development which deserves a place in academic curriculum, then we should rely on new relevant publications from academic libraries and anticipate that we will be no texts books in future.

## POTENTIAL SOLUTION TO THE PROBLEM

There are known recommendation in learning theories which could be used here to address the problem defined in the previous section and they are easy to see. We must avoid information overload in the learning process, eliminate any possibility of over-assessing students, and create the environment in which students could be comfortable to learn without a prescribed textbook.

The following was used:

The first step was to make a synergy between software development and SA by scaling down both disciplines into "basic principles" in order to allow students to learn and apply results of learning. Therefore, only component based and layered architectural style was used and UML modeling concepts of use cases with sequence diagrams were explored.

The second step was to expose students to academic source of materials which can be used in learning and promote research and exploration. This would address the lack of textbooks and teach students that their learning in the world of software development will continue even after they get their first jobs. Examples of software architectures in conjunction to UML modeling were sourced from the IEEE publications created form USN research and students MSc and BSc projects.

The third step was to scale down the classroom in small groups of students within which the learning could happen. This was the only way of measuring students learning curve from week to week. It was also an opportunity to address differences in learning and answer questions individuals may have. Consequently, formal lectures were used as "guidelines" and not as a source of knowledge which could be assessed in any type of assessment.

The fourth step was to make sure that the practical workshops focus on explorative learning and debates at the group level. However, the complexity of the material did not allow for "walking through many examples". Overloading student sin mere 12 weeks with a full scale of real life examples would be counter-productive and the learning would not happen. Instead of this, 12 different examples were created to play the role of home assignments, in which each group could exercises (A) real life preparation for software modelling, and (B) definition of component based and layered architectural style for the chosen problem. The examples ranged from automation in traffics and in modern cars, managing traffic congestions and autonomous busses, to flagging fake content of webpages, systemizing conflicting information during covid pandemic, and creating software solutions for forestation using drones. Both assignments were used as a check point for students' learning and confirmation that students will be able to cope with the formal examination.

Finally, there was no formal submission of the home assignments. The requirement was that each group debates with the tutor the problems they experience when working on the home assignments and possibly answer questions the tutor may have had. These debates did not have time limit. It was important that each group takes as much time as needed to master home assignments and feel confident that they can take the exam.

Unfortunately, we did not have time to have public presentation of all home assignment and facilitate knowledge sharing. The difference between software models and architectures, produced by different groups, were striking and thus the sharing of these models publicly, through presentations and debate, is very important for a healthy learning curve for each individual student.

## RESULTS

If we ignore problems created by the lack of students' attendance on occasions and the impact of the pandemic on the academic year, the students proved to be extremely successful in gaining essential knowledge on how to approach modeling with abstractions and how to conceptualize these models in SA. There were at least 6 groups (out of 9) which created a perfect full scale and commercially available models of their software solution. Some of them had models which exceeded the level of expertise we expect from BSc students. From this perspective, the subject proved to be deliverable within the curriculum.

However, there were a couple of problems which were not expected, and which triggered questions, already formulated in the abstract: "how should we really address pervasive computing in modern curriculum".

The first problem was related to students' difficulties to think independently. They still favored situations in which they repeat "knowledge" delivered by the tutor, try to memorize as much as possible and depend on written materials which solely support "tutor's words".

Secondly, their learning was constantly interrupted with their worries on "what will be in the exam" and asking for past exam papers, almost from the beginning of the semester. Considering that the subject ran for the first time in our new program, it was impossible to reassure students at the beginning of teaching, that the role of the tutor is to make sure that they are ready for the formal exam.

Thirdly, students were not keen on answering questions such as "what do you think" or "what would you like to do". Insecurities in saying "this is my opinion … because…" was striking in spite of clear evidence that most of the students mastered complex and real-life problems of software development in their home assignments.

Forth, the attempt to create exclusively student-centered learning, in which students "solutions to the problem" is NOT juxtaposed to any other solution, even to the tutor's solution, did not help students to understand that they are creators of software solutions, and in this process no-one anyone outside the group, including the tutor, ***have no say***. Students were still very much relaying on the judgmental approach in learning and expected "black and white answers on what is right and wrong" in software modelling. When we deal with abstractions and human perception in software modelling, there are no wrong and right answers.

Where do we go from here?

## CONCLUSIONS

The example described in this paper shows that it is feasible to create one subject “out of line” with the rest of the program which

a) addresses the latest changes in computing, software and communication technologies

b) introduces the characteristics of pervasive computing into the mainstream program and

c) adheres to the regulated education requirements in terms of having learning outcomes mapped to the assessment.

However, despite the proof that we can go ahead with having a *“flexible and changeable subject(s)”* to address the demand imposed by changes in technologies, this is just a modest attempt to address bullet points from the Introduction. It is almost impossible to address them completely, without looking at the goals of our modern programs in computer science and software / computer engineering, changes in teaching and learning practices and debating on exactly how we will measure student learning curves in future.

This paper might open the debate on the future of our BSc courses in the domain of computer science, computer and software engineering and pervasive computing. It can affect many other disciplines such as engineering, automation, business/management, socio-technical systems and similar. Are we ready for making changes?

It is very difficult to recommend future works. We itemize choices the USN may have. They must be debated across departments which deliver the current program, and focus on:

- Creating a flexible and adaptable curriculum, where major revision is incorporated in the goals of the program. This will secure constant alignment with advances in technologies and engineering,
- Creating specific pathways within the program, which could have separate specialization and address specificities of future changes in computer science and technologies,
- Resourcing subjects adequately and revisit all 5 credit subjects (does the learning really happen in these subjects?),
- Revisiting the program and decide on what must be sacrificed: this is a computer engineering course, and it does not have to embark on computer science and software engineering (pervasiveness can be addressed through advances in engineering),
- Moving towards computer science academic programs, because pervasiveness in our modern world (and in engineering) is solely addressed though computer science paradigms.

## REFERENCES

L. Bass., P. Clements, R. Kazman, (2021) Software Architecture In Practice (SEI Series In Software Engineering), Addison-Wesley Professional; 4th Edition (7 Oct. 2021)

Q. Burke, C. Sunrise Bailey (2020) Becoming an 'Adaptive' Expert, Communications of the ACM, September 2020, Vol. 63 No. 9, Pages 56-64

J. Lockwood and A. Mooney (2017) Computational thinking in education: Where does it fit?‖ International Journal of Computer Science Education in Schools, vol. 2, no. 1, pp. 1-20, 2017.

J. Lockwood and A. Mooney (2017) Computational thinking in education: Where does it fit?‖ International Journal of Computer Science Education in Schools, vol. 2, no. 1, pp. 1-20, 2017.

OMG (2004) UML Specification, available at https://www.omg.org/spec/UML/2.5.1/About-UML/

N. D. Saidin, F. Khalid, R. Martin, Y. Kuppusamy, and N. A. Munusamy (2021) Benefits and Challenges of Applying Computational Thinking in Education, International Journal of Information and Education Technology, vol. 11, no. 5, pp. 248-254, 2021.

M. Tissenbaum and A. Ottenbreit-Leftwich. 2020. A vision of K-2 computer science education for 2030. Commun. ACM 63, 5 (May 2020), 42–44.