ADDING CDIO-COMPONENTS TO (NON-)CDIO COURSES

Christian W Probst

DTU Compute, Technical University of Denmark

ABSTRACT

CDIO projects teach engineering students relevant competences and improve the learning process by integrating topics within and across semesters. These benefits often seem unachievable for individual courses that are not embedded in a CDIO setting. In this article we present an approach to achieve CDIO-like effects in non-CDIO courses, also outside of engineering curricula.
In typical project-based courses, students work on one or more individual phases of a project. The bigger the project in question, the bigger the efforts to fit the whole project inside a course, and the steeper the learning curve for the students.
Our approach personalises projects with extensions based on the students' experience and level of knowledge and competencies. In the beginning of the course, students get a functional subset of the project to work on. During the semester, they extend the project subset through a set of assignments. These extensions are partly mandatory, partly either chosen by students from a set of well-known, pre-specified extensions, or suggested by students. To ensure that each group reaches a minimum level of experience, each extension is assigned a value; each group has to choose extensions with a sum value that passes a certain threshold. For extensions suggested by groups, the group members and the educator determine a value. The resulting project setup enables groups on all levels to adapt their project to their capabilities, knowledge, competencies, and, last but not least, level of ambition.
We have implemented our approach in a compiler course on the 3rd term of our CDIO-based B.Eng. study line, and the results are very promising. Students appreciate the possibility to work on open problems, and the educators benefit from the kind of closed setting the projects run in. The project is based on a compiler for a subset of the programming language Java. The students have sufficient experience to know some standard extensions that are missing in the initial project, as well as other concepts that are not present in Java, or are not supported by the course project. This leads to interesting and challenging nuances for some of the groups.

KEYWORDS

Computer Science, project work, Standards 2, 4, 5, 7, 8, 9, 11

INTRODUCTION

CDIO projects have proven to teach engineering students relevant competences and to improve the learning process by integrating topics within and across terms. These benefits often seem unachievable for individual courses that are not embedded in a CDIO setting, or that have not yet been transformed to comply with the CDIO standards.

In this article we present an approach to add CDIO-like elements to project-based courses, be they CDIO-based or not. Using our approach, educators can achieve CDIO-like effects in non-CDIO courses, also outside of engineering curricula.

Engineering courses, especially in CDIO-based curricula, often are centred around projects that students work on in groups. Depending on the size of the problem, the efforts to fit the whole project inside a course can be rather big, and the learning curve for the students quite steep. Therefore, especially in early semesters, the projects are closed problems with a predefined methodology and solution. For many students, this is the appropriate approach. However, in every student population there is a group of students that will enjoy working on open problems from the first day.

Our approach addresses all these issues, independent of the status of adaptation of CDIO standards and branch of study. The only requirement is for the course work to be based on some kind of project that the students extend during the course. Our approach works by personalising this project and its extensions based on the students' experience and level of knowledge and competencies. This personalisation can either be implemented by the educator, or by the students, or in mutual agreement.

In the beginning of the course, or at the beginning of a project work period, students get a functional subset of the project to work on, and to through a set of assignments during the semester. The kind of extensions and assignments are not constrained. There can be any number or combination of mandatory, optional, and elective assignments, and they can be predefined, chosen by students from a set of well-known, pre-specified extensions, or they can be suggested by students. The spectrum from completely predefined by the educator to completely suggested by students results in fine-grained customizability of the project. Depending on characteristics of the class or the groups – for example, their level of knowledge, competencies, or ambition – project and extensions can be fine-tuned to match the students' requirements.

To ensure that each group reaches a minimum level of experience, each extension is assigned a value. For extensions suggested by groups, the group members and the educator determine a value together. This step is important to ensure that trivial or unreasonably complex solutions are excluded, and that students later will have the feeling of a point value in line with their work efforts. The resulting project setup enables groups on all levels to adapt their project to their capabilities, knowledge, competencies, and, last but not least, level of ambition.

The remainder of this article is structured as follows. After a description of the course, in which we have developed this approach, and the study line, in which the course is embedded, we discuss how to add CDIO components to project-based courses. We then highlight some of the challenges faced by educators and teaching assistants, and briefly discuss how to apply our approach to courses in non-engineering curricula.

COURSE DESCRIPTION

Before discussing our approach in the next section, we briefly present the course, in which we have developed our approach: a compiler course for 3rd term B.Eng. students. The course is embedded into two study lines: one for software technology, and one for computer systems engineering. Both study lines have recently (Nyborg, Probst, & Stassen, 2015) been developed from a common predecessor (Sparsø, et al., 2011), to strengthen the focus on software and

computer systems engineering, respectively. We therefore only describe the software technology study line.

Study Line "Software Technology"

As mentioned above, the course in which we have implemented our approach is part of a B.Eng. study line on software technology. The study line covers software engineering and core computer science, with a focus on systematic approaches for requirements engineering, design of system models, system implementation, and finally the deployment of the system, thus clearly mapping to the core competencies in the CDIO syllabus (Crawley, Malmqvist, Östlund, & Brodeur, 2007).

Table **1** shows the study plan for the respective study line, which is a development of an earlier B.Eng. study line on computer systems engineering (Sparsø, et al., 2011) as part of a merger between the Engineering College Copenhagen and the Technical University of Denmark (Nyborg, Probst, & Stassen, 2015). Throughout the first 4 obligatory terms, students get a basic education in software engineering, systematic development, and mathematics, eventually leading up to the CDIO project in the fourth term.

In each term several courses contribute to a (smaller) CDIO project. In the computer systems engineering version of this study line, the compiler course contributed to the 3rd term CDIO project together with a course on hardware/software (Todirica & Probst, 2014). With the revision of the study line as part of the merger, this collaboration was no longer possible, leading to the need for adding more CDIO aspects to the now stand-alone project in the course.

The Compiler Construction Course

The overall aim in the 3rd term course "Compiler Construction" and the associated project is to teach the fundamentals of translating high-level languages into native code, and how a compiler realizes this task, with a focus on the software side of this process. During the semester, the course teaches scanning, parsing, and intermediate representations (2 weeks), semantic analysis (4 weeks), processors, virtual machines, and runtime environments (2 weeks), and code generation (3 weeks).

Table 1. Study plan of the study line B.Eng. Software Technology.

Term	5 ECTS	5 ECTS	5 ECTS	5 ECTS	5 ECTS	5 ECTS
1	Mathematics	Discrete Mathematics	Version Control & Testing	Development methods for IT systems	Introductory Programming	
2	Database Programming	Algorithms and Data Structures	Data Communication	Networking Lab	Advanced Programming	
3	Probability and Statistics	OOAD	Game Physics	Compilers	Development for Mobile Devices	
4	Operating Systems	Distributed Systems	Parallel Systems	Model-based Software Development	CDIO Project	
5	Electives				Innovation	
6	Internship					
7	Electives		Exam Project			

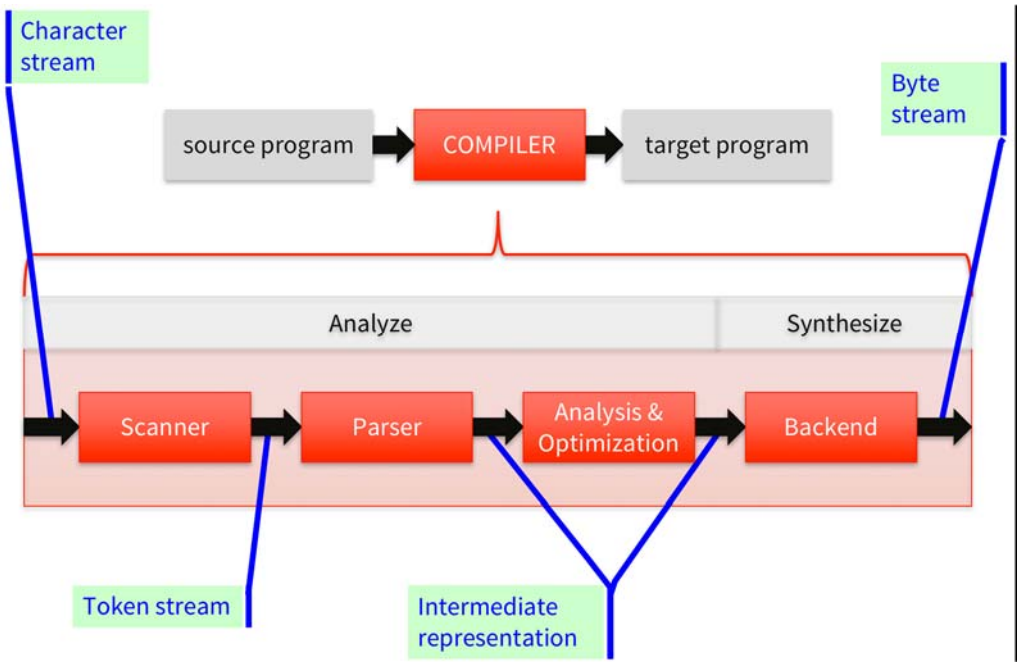

Figure 1. Structure of a compiler. The course covers all four phases (scanner, parser, analysis, and backend).

In the project associated with the course, students get a rudimentary, working compiler for a subset for Java (Appel, 2002). The task is to add functionality to the phases of the compiler, shown in Figure 1, by adding support for new constructs from scanning to code generation. When the compiler course still contributed to the 3rd term CDIO project in the earlier version of the study line (Todirica & Probst, 2014), this training contributed to the semester project, as the students had had intensive training in adding functionality, which they could use in the project as well. Interestingly, the decoupling of the two courses as part of the re-design of the study lines described above, resulted in a clearer project for the compiler course.

The learning objectives for this course are

- Understand the principles of compilers and virtual machines.
- Use and construct software tools to implement a working compiler.
- Explain the different phases in compilation and execution;
- Operate selected tools relating to the compiler phases (e.g., lexers, parsers);
- Explain the different elements in the description of a programming language;
- Derive specifications of the compiler phases, given a textual description of the syntax of a programming language;
- Implement an analysis and code generation phase, given a textual description of the semantics of a programming language;
- Acquire necessary skills to navigate large code bases; and
- Develop a working compiler.

The focus of the course is on understanding the rather large code base of the initial subset of the working compiler, extend it with new functionalities, and do so through all phases of the compiler. Based on their earlier knowledge and skills, students are able to transfer that knowledge to the implementation of new elements.

ADDING CDIO TO PROJECT-BASED COURSES

We now describe how the original, pre-CDIO version of the compiler course was adapted to embed a CDIO-like project in the course. In the next section we will then discuss how this approach can be generalised to other courses and even non-engineering curricula, and consider implications for educators and teaching assistants.

Our approach personalises projects with extensions based on the students' experience, knowledge, and competencies. In the beginning of the course, students get a working prototype of the project to work on. During the semester, they extend this prototype through a set of assignments. These extensions are partly mandatory, partly chosen by students from a set of well-known, pre-specified extensions, and partly suggested by students. To ensure that each group reaches a minimum level of experience, each extension is assigned a value; each group has to choose extensions with a sum value that passes a certain threshold. For extensions suggested by groups, the group members and the educator determine a value. The resulting project setup enables groups on all levels to adapt their project to their capabilities and level of ambition.

The Project

At the beginning of the course, students get a rudimentary, working compiler for a subset of Java (Appel, 2002). Since the programming education in the study line is based on Java in the first four, mandatory terms, students know the programming language and many of its constructs sufficiently well to be able to either map new constructs to existing ones, or to develop approaches to identify the new features. The task in the term-long project is to add functionality to the individual phases of the compiler, as they are being covered in the course.

Programming a compiler is a daunting task, and the initial project is of considerable size. The goal of the project is thus not only to give students the opportunity to develop a compiler – it is also to train them in navigating and exploring large code bases, a skill that will be essential when they start working, and that repeatedly has been demanded by our industrial advisory panel.

Throughout the term the students therefore get two kinds of assignments: one about extending the compiler, which we discuss in more detail in this paper, and one about exploring, understanding, and explaining the project that is handed out. These activities provide the students with the insight into the structure and the functioning of the project that they need for extending it.

We use the term "project" for the semester-long activity in the course. This overall project is the task of implementing extensions to the compiler. The individual extensions, which will be covered in the next sections, can be seen as group exercises. However, most extensions need to be considered in all phases of the compiler (see Figure 1), so students are repeatedly exposed to the problem how a certain programming language extension is added to the different phases of the compiler.

The required effort for the extensions can differ a lot; some may only require small changes in early phases, others may require changes to several phases, and finally some require quite

Table 2. Menu of extensions for students to choose from.

#	Description	Value
1	datatype char	5
2	Implement square root as call to Math.sqrt()	10
3	datatype double	10
4	pre or post increment/decrement (x++)	10
5	private	10
6	one other operator, eg, greater than, divide, modulus, exponent	10
7	ternary operator (a ? b : c)	15
8	for-loops	15
9	untyped variables (var x)	20
10	constructors	20
11	type cast	20
12	exceptions and try, catch	30
13	SQL support a la LINQ	60

involved changes of large parts of the compiler. While implementing these changes, and learning about the structure of the compiler code, students not only acquire the skills to implement a compiler, but they also get a better understanding for, how the Java language works internally.

Group forming

We briefly want to mention one change we have applied in the course, which is independent of the main topic of this paper. We asked students to register their level of ambition among the choices “top grade”, “medium grade”, “just pass”. We have then asked them to form groups with peers with a matching level of ambition. This small change has reduced the number of group changes in the term, and frustration between the students, significantly.

Student teams, which have formed over the first and second semester, usually have the same level of ambition anyway, but for students without a fixed group or exchange students this change has made group forming much simpler.

Predefined Extensions

Once students have formed groups and have started to explore the code base, their first task is to choose the extensions they will add to the compiler project. One of the first assignments is therefore to pick extensions from the list shown in Table **2**. To ensure that each group reaches a minimum level of experience, each extension is assigned a value. In our course, students must choose extensions of at least a given sum value, allowing them to choose from the simpler ones, but also challenging them to choose some of the more complicated ones.

The value of extensions shown in Table 2 represents approximately the difficulty or workload of adding the extension to the project. Adding the data type “char” to the project, for example, is considered to be relatively easy, since the initial compiler already contains a data type “int”. On the other hand, adding support for exceptions or inner classes requires significantly more work.

Defining the extensions requires some attention to the detail. It is important that the values of extensions “seem right”, that is fit with the students’ understanding of how difficult it is to implement them. A discussion in class about the extensions and their values is an excellent opportunity to give students an initial idea of, what kind of work is needed for some of the extensions. This is also a good preparation for students to choose extensions and to define their own extensions.

The predefined extensions mostly represent bucket-of-water projects (Vigild, et al., 2009) with a fixed starting point and a fixed result, but an open method: when implementing the extensions, students are free to choose a method to do so, as long as the result fits into the compiler. By providing a functional subset of the project, students can explore how the features present in this subset are implemented, thereby “learning from the known”. This decoding of available information and transferring it to a new problem is an enabling factor in our project work. Through the study of the existing project, students are encouraged to understand certain aspects of a problem. In a next step they realise that the same technique can be applied to similar problems.

Student-defined Extensions

For many students, picking only from the suggested, predefined extensions is sufficient. On the other hand, many students prefer open projects that enable them to explore new ideas, and to challenge their knowledge and competencies. Extensions to a predefined project combine a staged setting, where the boundaries of what students will work on are known, with an open setting, where students can define themselves what they will work on.

Table **2** shows some of the extensions suggested by students and the values assigned to them. For student-defined extensions, fixing the value is an important part of the process. The group members and the educator determine a value together. This step is important to ensure that trivial or unreasonably complex solutions are excluded, and that students later will have the feeling of a point value in line with their work efforts. In general, it will be difficult for students to judge the effort required for implementing an extension; this is especially true for the required work in later phases in the compiler project, which they might not even be aware of yet. On the other hand, students tend to overestimate how many points they should receive for their suggestions.

Table 2. Menu of extensions suggested by students.

#	**Description**	**Value**
S1	variable declaration with initialiser (int x = 10;)	15
S2	static array initialisation (int[] x = 1, 2, 3;)	15
S3	combination of operator and assignment (x+=7;)	15
S4	declare variables in the middle of the code	15
S5	struct and typedef	25
S6	inner classes	30

Evaluation

The reception by the students has been mixed in the first attempts, but the overall trend is very positive. While some of the students struggle with the potential choice of self-defined extensions and choose only from the pre-defined ones, a large group of students appreciates the possibility to work on self-defined, open problems, and makes eager use of this possibility together with sparring with their peers and the educators. It is clear from the discussion that they advance well beyond the level of understanding that has been observed in earlier editions of the course.

The evaluation of the course is currently a combination of project work and written exam. To reduce the potentially negative impact of a failed project, the project part contributed a smaller percentage to the grade than it did in earlier years. Nevertheless, the grade average has been stable over the last years, indicating that our approach has contributed to a significantly better understanding of the course's topic. This observation is supported by an evaluation of the answers in the written exam.

CHALLENGES FOR EDUCATORS AND TEACHING ASSISTANTS

The potential degree of free choice in the project is not only attractive to students, but also to educators and teaching assistants. Since students, who contribute their own extensions, can be expected to be highly motivated, the interactions with them very likely are rewarding for all parties. The focus shifts from repeating known facts to enabling innovative processes.

On the other hand, the amount of freedom also can result in some challenges, which need to be addressed. First of all, as mentioned above, the student-defined extensions need to receive a value that represents the amount of work required to add it to the project. This valuation needs to be realistic and adequate to avoid frustration, and it needs to be communicated in a clear way to students to avoid any misunderstanding. This requires a sufficiently deep understanding of the project and possible extensions by the educator, such that trivial or too difficult solutions can be avoided. Difficult extensions are in principle acceptable, if the educator and the students both are aware of this; in that case the students can learn even more about the topic of the project.

The amount of freedom requires special effort for the supervision of the personalised projects. For a staged project, solutions can be prepared or are even available before the start of the term; for the student-base extensions, this is naturally note the case, at least not for completely new extensions. While educators and teaching assistants might have an idea about feasibility, required effort, and potential obstacles, preparing a solution requires intensive work, which ideally can be executed together with the students who suggested the extension.

The challenges of freedom of choice also applies to the evaluation of the project work. How should a group that tries to solve a very advanced problem be evaluated in comparison to a group that only solves standard extensions, for which a solution can be found in the source code? Failing to solve advanced problems can have many different reasons, from very subtle challenges to lack of competencies. Ideally, the examination in this setting would be oral; due to the large number of students taking this course, this is currently not feasible. Instead, students get continuous feedback on the individual phases.

CDIO-FYING GENERAL COURSES

The applicability of the approach described above to CDIO course is obvious. However, it is equally well suited for all kinds of project-based courses. The only requirement is that the project can be structured in a way that students obtain a subset of the project in the beginning of the class, and that they work on this project throughout a certain period.

Any project that can be structured in an initial, functional subset and extensions, which are either pre-defined or student-defined is suitable. As pointed out above, it is important that the extensions are valuated beforehand, and that a risk assessment for student-defined extensions is performed to judge their chance of success and the required amount of work. It is of uttermost importance that students are protected against or at least receive an advance notice of potentially prohibitively complex extensions, both to avoid failure and to avoid frustration.

While we have developed our approach in a computer science course, and have discussed it with colleagues within engineering sciences, the approach is equally applicable to non-engineering curricula and courses. With a colleague we are currently working on applying the approach to a psychology course; here, the “functional subset” is the description of a case study, and the pre-defined extensions are aspects of the case study that students are asked to investigate. We envision the student-defined extensions to be further aspects of the case study, or different methodologies to apply. The whole project will be purely report-based, as a kind of gedankenexperiment.

For courses included in a CDIO curriculum, our approach contributes to a number of standards, mainly those related to learning and learning assessment, but also to introduction to engineering, design-implement experiences, and enhancement of faculty competence. We believe that the latter is especially interesting, since it reflects the educator’s involvement in enabling unpredictable student-defined extensions and their implementation.

CONCLUSION AND FUTURE WORK

In this article we describe how to add CDIO aspects to courses in CDIO study lines, but also to courses (still) outside a CDIO curriculum or during introducing CDIO. We argue however that our approach is so general that it equally well can be used in study lines that are outside the engineering or natural sciences. We are currently exploring options for applying our approach to a course in the social sciences.

The personalisation of projects for individual students or groups of students has proven to be an excellent factor for unleashing students’ creativity, and for giving them freedom even in almost staged projects. By being able to either choose a set of predefined extensions or by suggesting their own extensions, students can be integrated in the project. The discussions about feasibility of extensions, the involved work, and the valuation of such extensions, contribute massively to the students’ understanding and learning.

As described above, another important factor is the availability of the functional subset of the project, where “functional” addresses very different aspects depending of the course’s topic. We are currently investigating the applicability of our approach to non-engineering curricula, such as the psychology class discussed above. We believe that CDIO-like structures can be developed in almost all sciences, with similar effects.

BIBLIOGRAPHY

Appel, A. (2002). *Modern Compiler Implementation in Java* (2nd Edition ed.). Cambridge University Press.

Crawley, E., Malmqvist, J., Östlund, S., & Brodeur, D. (2007). *Rethinking Engineering Education. The CDIO Approach.* New York: Springer.

Nyborg, M., Probst, C. W., & Stassen, F. (2015). Developing merged CDIO based curricula for diploma (B.Eng.) IT study programs at DTU. *Proceedings of the 11th International CDIO Conference.* Chengdu, China.

Sparsø, J., Bolander, T., Fischer, P., Høgh, S., Nyborg, M., Probst, C. W., & Todirica, E. (2011). CDIO projects in DTU's B.Eng. in IT study program. *Proceedings of the 7th International CDIO Conference.* Kgs Lyngby, Denmark.

Todirica, E., & Probst, C. W. (2014). Mind the gap - teaching computer and compiler design in lockstep. *Proceedings of the 10th International CDIO Conference.* Barcelona, Spain.

Vigild, M. E., Willumsen, L. E., Borchersen, E., Clement, K., Bjerregaard Jensen, L., Kjærgaard, C., Klit, P., & Sparsø, J. (2009). Comparison and Classification of Design Build Projects in Different Engineering Bachelor Programs. *Proceedings of the 5th International CDIO Conference*, (pp. 1-7). Singapore.

BIOGRAPHICAL INFORMATION

Christian W Probst is an Associate Professor in the Department of Applied Mathematics and Computer Science at the Technical University of Denmark, and director of studies of a B.Eng. study line Software Technology. His current research focuses on organizational security as well as embedded systems and compilers.

Corresponding author

Christian W Probst
Technical University of Denmark
DTU Compute
DK-2800 Kongens Lyngby
+45 45 25 75 12
cwpr@dtu.dk