TESTING AND DEPLOYMENT OF SOFTWARE SYSTEMS (IN PRACTICE)

Mads Nyborg, Stig Høgh

DTU Compute, Technical University of Denmark

ABSTRACT

The CDIO concept is now well integrated into many curricula at universities around the world and it has meant an increase in the quality of engineering education.

However, the main focus has been on design-build projects and less on the 'C' and 'O' part.

In particular, the 'O' part of CDIO has received very little focus, since this is probably the most difficult part to implement in a university environment.

Because of this observation, in 2011 we decided to launch a new elective course, 'Testing and deployment of software systems (in practice)', focusing entirely on the 'O' part in CDIO.

The aim of this paper is to describe:

- the unified software development process and compare this with CDIO.
- the activities covering the 'O' part in software engineering.
- the course structure and schedule.
- the evaluations and comments received from students.

The paper concludes that:

It is possible to give students a realistic experience of the 'O' phase of CDIO. The prerequisite for this is that the course's entry level is a working prototype.

It is very important to identify an actor outside the university, which can act as a client (customer). This gives the students a more realistic environment.

The course also prepares the students for meetings with industry, taking place in the 6th semester, during the students' internship and later in the exam project in the 7th semester.

KEYWORDS

CDIO-based study programs, Testing, Deployment, Operate, Industry contacts (to university), product maturity, pre-internship, Standards: 1, 2, 7, 8.

INTRODUCTION

In September 2008, DTU's B.Eng. study programs were changed so that they are now based on the CDIO concept. For most of the study programs, this change has called for significant revisions. The CDIO standards form the basis for the new study plans. It was decided to introduce four cross-disciplinary 'CDIO' projects on each of the first four semesters. For the B.Eng. IT study plan, these four cross-disciplinary projects have replaced 11 smaller, course-specific projects in the old study plan. (Sparsø et al., 2007)

All the B.Eng. study programs follow a common structure. On the first four semesters, all courses are compulsory. With some variation, this is followed by a fifth semester with elective courses, a sixth semester with industry internship and a seventh semester with the final B.Eng. thesis. (Nyborg et al., 2012)

Each semester consists of a lecture period (13 weeks) and a lab-period (three weeks). All the courses involving CDIO semester projects are worth 10 ECTS points for both the lecture period and the lab period. (Sparsø et al., 2011)

On the first three semesters, the projects are design-build projects. The projects are attached to a 10-ECTS-point course with contributions from one or more supporting courses. For an example, see Nyborg & Høgh (2010). On the fourth semester, the project is a stand-alone project that also covers the 'C' and briefly touches the 'O' part.

The curriculum structure is shown in Table 1. The boxes marked in red indicate courses that contribute to the semester CDIO project.

Table 1: Study plan for the B.Eng. in IT study.

	Semester period (13 weeks)					**Lab period (3 weeks)**
	5 ECTS	**5 ECTS**	**5 ECTS**	**5 ECTS**	**5 ECTS**	**5 ECTS**
1	01906 Math. 1	01917 Math. 2	31021 Electronics	02313 Dev. methods for IT systems	02312 Introductory programming	
2	01917 Discrete math. and databases	02311 Digital Systems	02325 Data communication	02326 Algorithms and data structures	02324 Advanced programming	
3	02323 Probability and statistics	02344 Objected oriented analysis and design (OOAD) and databases		02332 Compiler construction	02321 Hardware / Software Programming	
4	02333 Parallel and real-time systems		02341 Model-based software engineering	02342 Distributed systems	02343 CDIO project	
5	Elective courses					
6	Engineering practice / industry internship.					
7	Elective courses		Final B.Eng. project			

The focus on all semesters is thus on the 'DI' part of CDIO.

In our opinion, the 'O' part of CDIO has seen very little in focus in the CDIO-based curriculum.

The simple reason for this is that there is not enough time in the semester projects. Moreover, the right environment may not be present in universities to explore this phase.

The outcomes of the semester projects are prototypes, which in almost all cases are not ready for deployment at the end-user.

Because of this observation, in 2011 we decided to launch a new elective course, 'Testing and deployment of software systems (in practice)', focusing entirely on the 'O' part in CDIO.

The course has now run twice and is the first course of its kind at DTU Compute.

The requirement for taking part in the course is that participants have a completed project. This could be a CDIO project, as mentioned above. However, it is more often a project/product completed in collaboration with an enterprise. This may be in connection with an internship, a final project, or a course carried out in collaboration with an enterprise.

Furthermore the students are asked to identify a customer or an organization outside the university which may act as a customer in a more realistic environment.

The course is now open to all students at DTU who have a completed a project.

We now have projects from other study directions, often projects that have been completed in collaboration with industry.

THE UNIFIED SOFTWARE DEVELOPMENT PROCESS AND CDIO

An important difference in developing software systems from other engineering disciplines is the recognition of the ever changing user requirements which, like in the waterfall model (Sommerville I., 2010), were unrecognized in the early development processes. During the 90s, alternative processes were developed that take into account this observation. These models are all based on incremental development where the user is in focus. In particular, the Unified Process (UP) (Booch et al., 1999) has been widely accepted in industry. This process is incremental and use-case driven. Use cases are a systematic way to describe functional requirements from a user perspective and are used from initial requirements-gathering to final testing of a system.

UP is used as the development process throughout the compulsory part of the diploma IT program.

UP is a two-dimensional development process containing *phases* and *activities.*

The process contains four *phases*: Inception - Elaboration - Construction – Transition.

Each of these phases contains the same six *activities*: Business modeling - Requirements - Analysis & design - Implementation - Test - Deployment, but with different focus, see Figure 1.

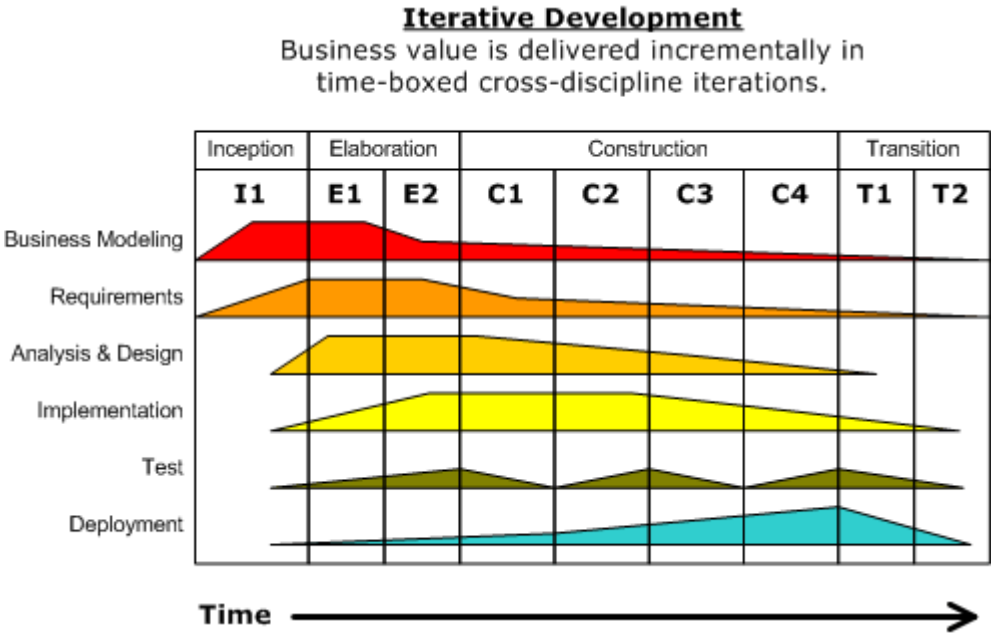

Figure 1. The Unified Process

The aim of the Inception phase is to gain an understanding of the domain and to build the business model, identifying risk and determining the scope of the proposed project. This phase contains preparation of a preliminary project schedule and cost estimate.

The Inception phase can be compared with the 'C' phase in CDIO, but it is not as comprehensive as this, since UP is a system development model, and does not include discovering unrecognized user needs.

The aim of the Elaboration phase is to address known risk factors and to establish and validate the system architecture. Common activities undertaken in this phase include the creation of use case diagrams, conceptual diagrams, design diagrams and architectural diagrams. Implementation of important use-cases ('proof of concept prototypes') are constructed. The Elaboration phase can be compared with the 'D' phase in CDIO.

The Construction phase can be compared with the 'I' phase in CDIO. In this phase the system is constructed in a series of *increments* (prototypes) illustrated with C1, C2,.. For each increment, tests are performed. Overall the testing activities are divided into two categories: *Black box testing* and *White box testing.* (Desikan S et al., 2006)

Black box testing (also called functional testing) is testing that ignores the internal mechanism of a system or component of the prototype and focuses solely on the outputs generated in response to selected inputs and execution conditions.

White box testing (also called structural testing) is testing that takes into account the internal mechanism of the components that constitutes the prototype in question. (Mohd et al., 2011)

The transition phase can be compared with the ‘O’ phase in CDIO, but it does not quite describe all the activities of the 'O' part, as it primarily focuses on deployment and user training. The tests performed in this phase refer to the user acceptance test which is described in the next section.

An important observation for software systems is that a prototype is generally not thrown away, but is further developed into a completed system deployed at the end user. This is illustrated in figure 1 by C1, C2,.. leading to an initial release of the system at T1 in the transition phase.

Throughout all testing cycles of the construction increments and final releases, regression test cases are run. Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements (Desikan S et al., 2006).

THE ‘O’ PART OF CDIO IN SOFTWARE ENGINEERING

All engineering disciplines contain an operate phase. In the CDIO context the operate phase uses the delivered, implemented product, process or system to satisfy the intended value; that is to satisfy the customer need. The operate phase includes maintaining, servicing, evolving, recycling and retiring the product.

In software engineering, in most cases the operate phase involves a transfer of the system from a development environment to a production environment.

The evolution of software systems over time is designated releases.

In UP, the operate phase is described by the transition phase and is also iterative in nature, starting with an initial release of the system and with each cycle controlled by new release versions of the system, illustrated with T1, T2 in Figure 1.

However, as described in the previous section, the transition phase in UP mostly focuses on deployment of the system and hence does not describe other important activities.

For each release of the system, the operate phase can be summarized by the following activities.

Release user acceptance test

Prior to deploying a new release, a release (including the initial release) user acceptance test should be performed. (Desikan et al., 2006)

User acceptance is a type of testing performed by the end-user to certify that the system fulfils the requirements that were initially agreed in the contract. This testing happens in the final phase of testing of a release before moving the application to the production environment. The test type is black-box testing.

From a contract viewpoint, this is important, since it is the transfer of ownership of the developed release to the end-user (customer).

Variations of the UP like agile development processes (Sommerville I., 2010) include acceptance tests in each construction increment (C1, C2,..), hence transferring ownership to the end-user in each increment. These processes were primarily introduced to ensure 'value for money' for the end-user and payment of work during development.

Preparing user documentation / manuals and training programs

Models and documentation of the system are usually prepared during the development process, but seldom end-user training material. This activity contains the preparation of user training manuals, training videos, FAQ and workshops.

Deploying the release

Most software system releases, except for cloud-based systems, are to be deployed in an operating environment at the end-user. This activity contains all operations to prepare a release for assembly and transfer to the customer site.

Important sub-activities include:

- Procedures for taking down the current release.
- Developing scripts for installing a release.
- Preparing installation manuals.
- Procedures for installing the new release.

Maintenance

Maintenance of software systems is concerned with modifications that do not involve developing new features, (Grubb et al., 2002)

The modifications can be categorized in:

- Corrective maintenance: Modification to correct discovered problems / faults.
- Adaptive maintenance: Modification to keep a software product usable in a changed or changing environment, e.g. moving the system to another hardware platform.
- Perfective maintenance: Modification to improve performance or maintainability.
- Preventive maintenance: Modification to detect and correct latent faults in the software product before they become effective faults.

In order to support maintenance activities, software systems in general should have a log feature built in that records usage of the different components and exceptions thrown by the system.

Also, a change log system should be provided to report faults within the current release.

Planning of new releases

On the basis of user experiences, new releases are planned. New features are identified from the end-users by using the current release. (Humble et al., 2010)

Major new features are in principle developed through the same phases as the initial release, i.e. inception, elaboration and construction. (Figure 1)

The feature wish list is usually prioritized in “need to have” and “nice to have” features.

To support the planning of new releases, a release planning system should be provided.

As described in the previous section, a regression test should be made before deploying a new release to ensure that the newly implemented features do not introduce new bugs.

This activity also contains decisions to either retire the system or to keep it as it is. In the latter case the system becomes a legacy system that can be used as a component in new releases. Usually, this decision is made for releases that provide useful functionality but are outdated from a technological viewpoint.

COURSE STRUCTURE AND PREREQUISITES

The purpose of the course is:

- to train participants in testing and launching a large complex program.
- to enable students to prepare and/or relate to requirement specifications for a large software project.
- to support the CDIO concept, with special focus on the Operate part.

In order to be able to participate in the course, the students must have access to a “finished” project.

This may be:

- Group work from a 10-ECTS-points course carried out in a previous study year.
- An individual exam project, typically carried out in collaboration with an enterprise (20-35 ECTS points).
- A project carried out in another context outside the university, typically in collaboration with an enterprise.

The overall requirement is that the project is a finished project which can be launched (Technical University of Denmark. Course description 02166).
At the beginning of the course, groups of 4-6 students are formed. One or more students can bring their finished project into the group.

Date of the course

Figure 2 shows the timeline of the courses compared to the standard timeline. The course begins at the end of the spring semester in the beginning of May. The course is followed by a three-week exam period. The students have other activities in this period, but there is time for a gentle start. A gentle start is important because contact with enterprises is often a lengthy

process. Then follows a period of three weeks of summer school (Sparsø et al., 2011). This is a period in which the students normally earn 5 ECTS points by working full-time.

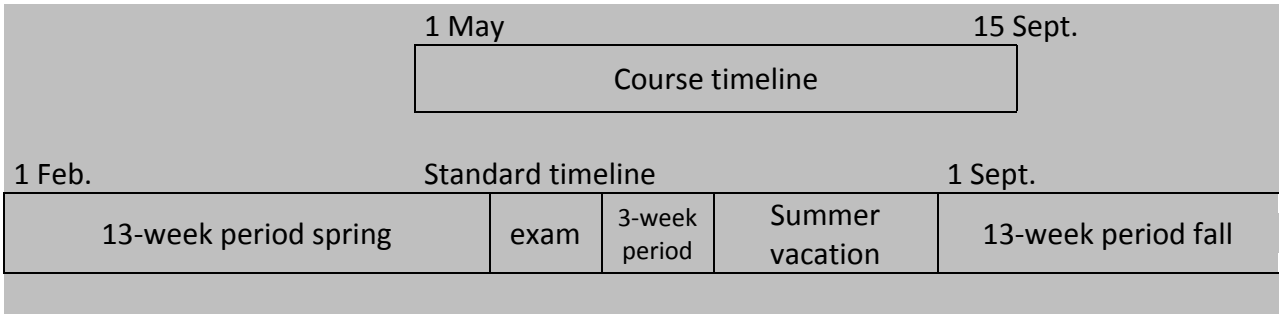

Figure 2. Course timeline

We have decided to lengthen the period so the students can work on their project during their summer vacation for six weeks and then two weeks into the fall semester.
This means that work is in low gear for 11 weeks. If the course had been held as a full-time course, it would have lasted for three weeks, corresponding to 5 ECTS points.
We held dialogue meetings with the students about once a week in the beginning of the process.

We discussed the individual projects in terms of:

- Scope and delimitation of the project.
- Expected level of completion.
- Necessary tools and appropriate work methods.
- Expectations of the enterprise.
- Expectations of the university.

As far as possible, these points will be discussed with the end-user/enterprise.
The meetings will continue during the entire process, however at longer intervals towards the end of the period.

Examples of projects

An XML-parser: Originally developed as an exam project in collaboration with a company. This project has been tested and implemented on the course, in collaboration with the company.

Crowd tracking system: Developed for use at Danish music festivals. Originates from an assignment completed by six students in a 10 ECTS points course. Two of these students have continued to mature the system as well as test and implement it at the Roskilde Festival.

Electronic voting system: Originally developed by a team of international students under an Erasmus IP project (Nyborg et al., 2011). The project was tested and implemented at the university. The product is now being used for voting on several teaching courses. Participants were students from the original project and new students.

Deposit system for returned bottles. Developed in collaboration with the Roskilde Festival. Originates from an assignment completed by six students in a 10 ECTS points course. The system was tested and implemented at the Roskilde Festival in 2011. Participants were students from the original project and new students.

Completion and evaluation

Each group will hand in a report at the end of the course period. Requirements for the report are in line with requirements the students are already familiar with from their master projects. Finally, exams will be held with external examiners.

The exam is composed of:

- Assessment of the report.
- Oral presentation of the report and product by the group.
- A possible poster which can be used to present/advertise the product.

Finally, all students are given an individual mark.

COURSE EVALUATION

At the university, all courses will be routinely evaluated centrally at the end of the teaching period. Due to the timing of this course, this has not been possible. Therefore, we have conducted our own questionnaire survey.
The target group was the students who completed the course in 2012 or 2013. We interviewed the students by telephone.
We then prepared a questionnaire. The questionnaire contained six questions and a free-text field. We asked the students the following questions:
Table 2 summarizes the replies to the six questions. Table 3 shows the free-text fields responses.

Table 2. Results of the questionnaire survey

On a scale from 1 to 5, where 1 means 'totally disagree' and five means 'fully agree'.		
1	I think I've learnt a lot on this course	4.1
2	I generally think the course was good	4.3
3	I think the course was relevant and it contributed with topics that are not covered elsewhere in the education	4.6
4	I think the teaching program invited me to be active in the course	4.4
5	Do you find that you need more time in the course, e.g. extend the course to 7.5 ECTS	3.1
6	I think the course should be mandatory in the curriculum	3.5

Table 3. The free text responses.

The students' feedback from the questionnaire (all three have been included uncritically).
"I think the course should be mandatory in the curriculum" - It depends on the course it is to replace, but it's basically a good idea. For example, the course could teach the students testing techniques for software: unit tests, integration tests and verification tests, as these are not taught in the mandatory subjects.
It could be interesting to see how the course would develop if the information meetings began already after 13-weeks so there was good time to find a product and a customers. There were already a couple of information meetings before the actual start of the course, and it would have been a good idea to contact some enterprises/customers who could be interested in the product. I don't think you would then need 7.5 ETCS points and commencement with 13 weeks period. On the question of whether the course should be mandatory, the score is medium. The course is very good and offers relevant information which is not offered by other courses. However, the course assumes that you already have some sort of product a customer could be interested in, and I don't think all students have that, and even though 1/5 will be "camp followers" of an idea, this is counterbalanced by the fact that it should be an idea/product a possible customer can approve.
Positive comments: + The best thing about the course was that we got to collaborate with an enterprise which could use the program we had worked on. This gave good (practical) experience which we otherwise possibly would not have had at the university (academic experience). + It was good to have an IT contract drawn up. It was good to experience this process and to look at the different aspects of the contract. We can definitely use this experience in the future. I think we should be introduced to this in our degree course because BSc - Engineering is a practical education program. Negative comments: - No specific books, tutorials, material or guidance on how to launch software systems. The students have practically no opportunities to relate to anything specific. In addition, there is a lack of input from the teacher. - Testing is an excessive part of the course. I think it's relevant in the course, but not to this extent. During the course, we learned about black-box tests, white-box tests, unit tests, etc. These tests should be made when developing the software product. I think this course should focus more on tests which are relevant for launching software, e.g. acceptance tests and similar.

Brief summary/keywords from telephone interviews with the students

Below we have made a summary of some of the strongest statements from the students. We have grouped the comments into four groups.

1. The project:

It feels good to get out in real life.
A generally good course, not so much unnecessary information, good to be under pressure from an end-user.
Super relevant to speak with a real customer / launch.
What should you do if you don't have a customer/end-user? I think it would be more difficult to run the course at the same level.
Very important to have a real customer. If this is not possible, you have to simulate one, but it will never be the same.
Getting out in real life. was great!
It was fantastic to be allowed to make a complete project.

2. Timeline of the course:

It was good to organize a long period of time myself.
The timeline is just fine.

3. Relevance for the degree course with regard to the labor market:

Relevance with regard to the labor market is definitely important.
Relevance with regard to the labor market: Extremely relevant.
It would have been nice with guest lecturers from industry.

4. Miscellaneous:

It should not be mandatory, because it's much more fun when it's voluntary.
Advertise the course more.
Version management would be relevant as a requirement/option.

CONCLUSION AND FINAL REMARKS

Due to the intangible nature of software systems, the time required for changes is often shorter than for other engineering disciplines. This means that, in one semester, it is realistic to be able to complete significant activities in the Operate phase as described in the previous section.

In order to maintain focus on the 'O' part, we have come to the conclusion that the starting point for this course should be a functional prototype. Moreover, it is important to have identified a player outside the university who can act as a customer.

When working with software, there is a realistic possibility to conduct all the aspects of the CDIO concept. Compared to other engineering disciplines, we have great advantages with regard to the accessibility of "real" tools within design and implementation.

In addition, financially it is often affordable to launch/test specific solutions. (the 'O' part of the CDIO). Most production of new versions/changes can be carried out without using significant physical material.

The course contributes to fulfill CDIO standards 1 (CDIO as Context) and 2 (CDIO Syllabus Outcomes) regarding the Operate part of the product's lifecycle.

Furthermore the course involves active communication with stakeholders outside the university, and much of the course is based on active learning. Therefore, it also contributes to CDIO standards 7 (Integrated Learning Experiences) and 8 (Active Learning)

We hope that this paper can help inspire more people to focus on the 'O' part of CDIO.

REFERENCES

Booch G., & Rumbaugh J., & Jacobsen I. (1999). The Unified Software Development Process: The Complete Guide to the Unified Process from the Original Designers. Addison-Wesley Publishing Company, 1999 ISBN 10: 0201571692, ISBN 13: 9780201571691.

Desikan S., & Ramesh R. (2006) Software Testing: Principles and Practice
Pearson Education India, Jan 1, 2006 - Computer software. ISBN 10: 817758121X, ISBN 13: 9788177581218.

E. Burt Swanson. (1976). The dimensions of maintenance. Proceedings of the 2nd international conference on Software engineering, San Francisco, 1976, pp 492 — 497.

Grubb P, & Takang A. (2002). Software Maintenance: Concepts and Practice. World Scientific Pub Co Inc; (2nd Edition), ISBN-10: 981238426X, ISBN-13: 978-9812384263.

Humble J., & Farley D. (2010) Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment. Addison-Wesley Professional; (1st Edition), ISBN-10: 0321601912, ISBN-13: 978-0321601919.

Mohd E. K. (2011). Different Approaches to White Box Testing Technique for Finding Errors. International Journal of Software Engineering and Its Applications, Vol. 5 No. 3, July 2011.

Nyborg M., & Høgh S., & Lauridsen P.(2012). Evaluation of the industrial internship for the Diploma IT programme at DTU. Proceedings of the 8th International CDIO Conference, Queensland University of Technology, Brisbane, Queensland, Australia, July 2nd to 4th, 2012.

Nyborg M., & Gustafsson F., & Christensen J. E. (2011). Developing Open Source System Expertise in Europe (DOSSEE). Proceedings of the 7th International CDIO Conference, Technical University of Denmark, Denmark, June 20-23, 2011.

Nyborg M., & Høgh S. (2010). Mapping an industrial IT project to a 2nd semester design-build project. (2010) Proceedings of the 6th International CDIO Conference, Ecole Polytechnique, Montreal, June 15-18, 2010.

Patton R. (2005). Software Testing. Sams Publishing (2nd Edition), ISBN-10: 0672327988
ISBN-13: 978-0672327988.

Sommerville I. (2010). Software Engineering. Addison-Wesley (9th Edition), ISBN-10: 013703515, ISBN-13: 978-0137035151.

Sparsø J., & Bolander T., & Fischer P., & Høgh S., & Nyborg M., & Probst C., & Todirica E. (2011). CDIO projects in DTU's B.Eng. in IT study program. Proceedings of the 7th International CDIO Conference, Technical University of Denmark, Denmark, June 20-23, 2011.

Sparsø J., & Klit P., & May M., Mohr G., & Vigild M.E. (2007), Towards CDIO-based B.Eng. studies at the Technical University of Denmark. Proceedings 3rd International CDIO Conference, 2007.
Technical University of Denmark. Course description 02166 Testing and deployment of software systems (in practice). http://www.kurser.dtu.dk/courses/02166/default.aspx

Biographical Information

Mads Nyborg is an associate professor in software engineering at DTU Compute. He has several years of experience in teaching in software engineering and has governed industrial projects both as a consultant and as a supervisor for student projects. He was the one of the primary movers in introducing the CDIO concept at the diploma programme at DTU Compute.

Stig Høgh is an associate professor in software engineering at DTU Compute. He has several years of experience in teaching software engineering and has governed industrial projects both as a consultant and as a supervisor for student projects. In the period 1985-2005 he produced software for quality control. He was one of the primary movers in introducing the CDIO concept at the diploma programme at DTU Compute.

Corresponding author

Mads Nyborg
Associate Professor
DTU Compute
Matematiktorvet
Building 303B
DK 2800 Lyngby
manyb@dtu.dk