A Software Configuration Management Course

Ulf Asklund, Lars Bendix

Department of Computer Science, Lund Institute of Technology, Box 118,
SE-221 00 Lund, Sweden
{ulf|bendix}@cs.lth.se

Abstract. Software Configuration Management has been a big success in research and creation of tools. There are also many vendors in the market of selling courses to companies. However, in the education sector Software Configuration Management has still not quite made it – at least not into the university curriculum. It is either not taught at all or is just a minor part of a general course in software engineering. In this paper, we report on our experience with giving a full course entirely dedicated to Software Configuration Management topics and start a discussion of what ideally should be the goal and contents of such a course.

1 Introduction

In the call for papers for the SCM workshop it says: "The discipline of software configuration management (SCM) provides one of the best success stories in the field of software engineering. With the availability of over a hundred commercial SCM systems that together form a billion dollar marketplace, and the explicit recognition of SCM by such standards as the CMM and ISO-9000, the discipline has established itself as one of the essential cornerstones of software engineering.".

The call furthermore says that "SCM is a well-established discipline". This, however, apparently does not apply to the educational aspects of SCM. To the best of our knowledge universities only teach SCM as a small part of a more general course in software engineering – or not at all. If one wants to teach a more extensive course on SCM topics, there is very little help to get with regards to the contents of such a course. Most books on the subject, like Berlack [10], Leon [24] and many more, all follow the traditional way of looking at SCM as consisting of "the four activities": configuration identification, configuration control, configuration status accounting and configuration audit. This more formal approach to SCM is, however, in our opinion an experience not quite suited for university students.

At Lund Institute of Technology, we have had the possibility to develop a full course dedicated entirely to SCM topics. In this paper, we want to report our experience from two years of teaching this course. We also want to start a discussion of what topics could be included in such a course and how SCM can be taught in a full course of its own as well as inside a software engineering course.

In the following, we will first describe how our course is given and what is in the course. Then we discuss the considerations that led to the contents that the course has today. This is followed by a discussion of the pedagogical considerations that we have had. Finally, we reflect on the experience we have gained from the first two years of giving the course, describe our plans for future changes to the course and draw our conclusions.

2 Description of the course

The course we give is at undergraduate level. The students are following the last years of their education and as a prerequisite for following this course they must have taken the project "Program development in groups" in which they in groups of 10 students during a period of 7 weeks develop a software product, thus facing many of the problems addressed by SCM.

What are the goals of the course? Given the amount of resources we have available for this dedicated course, we can be rather ambitious. We can prepare our students for many of the situations that they will meet in industry after they graduate. Most of our students will probably make a career in industry and from an SCM perspective the course should guarantee that they are qualified to cover positions ranging from developer through project leader to SCM expert or manager. The course is optional, so we can count on the students following it out of interest.

If we look at SCM, it is related to many different roles on a company-wide scale. We have distinguished at least the following roles: developers, SCM wizards, project leaders, SCM experts or managers, SCM tool administrators, quality assurance and test, release people, and SCM tool developers. An ambitious SCM course should consider the needs of all these roles. The goals for our course are to provide the developer with knowledge of traditional SCM, comprehension of problems and solutions in development in groups and application of work models including the use of functionality like merge and diff. In addition to what the developer learns, the SCM wizard on a project team should have an analytical insight into problems and solutions in SCM for groups and should reach application level for tasks that are more rarely performed, such as weekly builds, creating branches and doing releases. The project leader needs to have comprehension of the SCM problems and solutions encountered by developers and he should reach application of traditional SCM including CCBs and SCM plans. An SCM expert or manager should ideally know everything. In our course we want to bring the students to a level where they can analyse SCM problems directly related to developers, synthesise one or more possible solutions and evaluate these solutions. In the present course, we choose to leave out the roles of SCM tool administrator and SCM tool developer, the first being too tool specific and the latter too advanced. For the quality assurance and test roles, we consider their needs to be at a usage level and thus a sub-set of the developer's needs. Likewise we consider, in part, the needs of release people to be covered by what is taught for the SCM wizard role.

Our SCM course lasts 7 weeks. Each week we have two lectures, one exercise session and one session in the computer lab – each of the duration of two hours. Each week is entirely dedicated to a specific theme and starts with a lecture that gives the basics of the theme. Then there is an exercise session where the students, based on open questions, discuss the theme of the week in small groups of 3-4 students. This is followed by a second lecture on the theme of the week. This lecture starts with student presentations of their results from the previous exercise session, followed by a in-depth treatment of topics within the theme. Finally there is a session in the computer lab, where the students work in the same groups as during the exercise sessions. The computer labs do not follow the themes, but aim at giving practical hands-on experience with two tools, CVS and ClearCase. There are five computer labs and six exercise sessions, which gives a total of 50 hours for the whole course, not including the time the students use for preparation and reading literature.

The themes that are treated in the course are the following:

1. introduction, motivation and overview
2. collaboration (construction site)
3. workspace (study)
4. repository (library)
5. traditional configuration management
6. SCM relations to other domains
7. wrap up, summary and question hour

Theme 1 is introduction, motivation and overview. The first lecture introduces SCM and gives some example scenarios of program development to motivate the students and explain why SCM is important. The exercise session is used for team-building, forming the groups that will work together during the rest of the exercise sessions, during the computer labs, and for the examination. We also use the first exercise session to get information about the students' background and their expectations to the course. The second lecture gives an overview of SCM and the rest of the course. The literature used for this theme is chapters from Babich [6].

Themes 2, 3 and 4 all deal with what a developer needs from SCM and to describe these aspects we use three metaphors [9]: a construction site, a study, and a library. A developer needs to collaborate with others (a construction site), to create a workspace where he can work undisturbed (a study), and a place where he and others can store the results of their work (a library). Common for the exercise sessions during these three themes is that we also use the metaphors to facilitate the discussions and that each group has to produce one slide containing the most interesting/surprising result from their discussions to be presented (by them) and discussed briefly during the following lecture.

Theme 2 – the construction site: The first lecture is basics about co-ordination and communication. The exercise session focuses on the importance of planning, co-ordination and communication and the second lecture goes more in-depth with work models and how geographical distribution affects SCM. Literature used is excerpts from [25], parts of [2] and a chapter from [30].

Theme 3 – the study: The first lecture is basics about roles, versioning and workspaces. The exercise session discusses what a workspace should look like to make it

possible for the developer to get his work done. The in-depth lecture is about SCM models and merging. Literature is excerpts from [31], [7] and [19].

Theme 4 – the library: The basics here are repository structures, identification and history. In the exercise session students discuss possible solutions for versioning, branching, selection and representing dependencies. The in-depth lecture covers versioning models and branching patterns. Literature is excerpts from [21], [3] and [1].

Theme 5 deals with the traditional way of looking at SCM as consisting of "the four activities". The first lecture explains configuration identification and configuration control with its main emphasis on the change process and the role and functioning of the change control board (CCB). The second lecture covers the remaining activities of configuration status accounting and configuration audit. Furthermore this lecture covers CM plans and roles. This week's exercise session is not strictly connected to the theme, as the session comes between the two lectures. Exercises are more focused discussion questions than the previous weeks and relate to the identification, structuring and process aspects of themes 2-4 seen in the light of this week's first lecture. As literature we use chapters from [14] and excerpts from [24] and [13].

Theme 6 covers domains related to SCM. We treat product data management (PDM), open source software development (OSS), software architecture (SA), and the use of patterns for SCM. Literature is excerpts from [12] for patterns, [8] for SA, [4] for PDM, and [5] for OSS. During the exercise session this week, the students have to put the previous weeks' work together within the framework of a CM plan. Based on their discussions during the metaphor sessions and with the knowledge they now have about traditional SCM and the purpose and contents of CM plans, they have to construct fragments of a CM plan for an imaginary project. This has to be written up as a 2-3 pages essay that has to be handed in before the examination.

The final theme wraps up the whole course. The pieces that each theme constitutes are put together again to form a whole. We also look back on what we've been through and focus on the relationships there are between the themes instead of looking at them in isolation. The second lecture during the final week is an external lecture. The first time we had an industrial presentation and the second time a PhD defence. Literature is excerpts from [15] and [23].

The computer labs are not strictly connected to the themes followed by the lectures and exercise sessions. Starting week 2 through week 6 there are five labs covering practical work with the tools CVS (3 labs) and ClearCase (2 labs). Labs are a mixture of detailed guidance and open ended experiments. The students work in the same groups as for the exercise sessions.

In the first CVS-lab the students familiarise themselves with the tool. They set up the repository and use it for simple collaboration using turn-taking. In the second lab they continue to explore the support for parallel development and automatic merge. They also make a release that is used in the third lab. The last CVS-lab is focused on the use of branches for maintenance as well as for experiments/variants. Furthermore, they investigate the possibilities for creating awareness within a group of developers. Finally, they have the possibility use tkCVS to compare a graphical interface with a pure command-based tool.

During the ClearCase-labs the students go through roughly the same set of tasks as for the CVS-lab, but now getting experience with a different tool. They do not have to

set up the tool and the repository, but they do explore the VOB and the configuration specifications. They create views exploring different collaboration patterns to discover the flexibility and power of ClearCase – and the price paid. They also create a release and turn back to it to create a maintenance branch – and to merge that branch back into the main line of development.

In the week following the last lecture, we have the examination. It is an oral examination which uses the essay produced during the last exercise session, but the examination covers the whole course curriculum. It is carried out in the same groups that the students work in during exercises and labs, it lasts for 15-20 minutes pr. student and is a group discussion more than an interrogation. The students get individual grades.

3 Contents considerations

The past two years at Lund Institute of Technology, we have had the opportunity to teach a full course dedicated to SCM topics only. This course is an optional course for students in the final years of their masters education in computer science. Usually the problem is to choose what to put into – and especially what to leave out of – one or two lectures about SCM given as part of a more general course in software engineering. Faced with the luxury of a whole course of 50 hours of lectures, exercise and lab sessions we were unsure whether or not we could actually fill all this time with relevant topics. All our doubts proved to be unfounded – quite on the contrary we found such a wealth of topics in SCM that even with a full course we were faced with having to choose between what to include and what to leave out.

With regards to a curriculum for an SCM course we looked around for help, but found very little guidance. There are a number of papers on the concepts and principles of SCM starting with papers by Tichy [29] and Estublier [17] from the very first SCM workshop. These two papers deal with fundamentals like versioning, selection, configurations and builds. Things that are definitely central and should be covered by any kind of SCM course. In fact, these topics are an important part of the contents in our course being treated in themes 2, 3 and 4. However, we do not find the papers themselves suited as literature because of their orientation towards tools and functionality that was of interest for the papers' original audience. This is also the reason why we chose Babich [6], who gives more motivational examples and less technical detail, as literature for theme 1 "Overview and motivation". Later on Leblang [23] wrote an overview paper about the fundamentals of SCM. It treats roughly the same problems as [29] and [17], but sees SCM as part of an overall process and has more bias towards tasks than tool functionality. In fact, this paper has a more general audience and made it into the course literature – serving as the final look back tying things together. More recently Estublier [18] wrote another paper providing a roadmap of the results that had been obtained by the SCM research community at the turn of the millennium. It provides an even broader view by including explicitly also co-operative work and product data management. Topics that are treated in themes 2 and 6 respectively of our course. Because it is a roadmap, it does not go into any detail. Likewise it is more focused on research results than on the actual problems they try to solve. For these

reasons we did not find the paper itself suited as literature – though an excellent list of important topics.

We also looked at general software engineering textbooks like [26] and [28]. By nature such textbooks cannot cover everything in each of the topics they treat. Both books provide good overviews of formal as well as more informal aspects of SCM. Being more hands-on oriented they focus on the activities of configuration identification and configuration control for the formal part. The informal parts concerns versioning, releasing and building of systems. We have included both aspects in our course, theme 5 covering the formal aspects and themes 2, 3 and 4 the informal aspects. Most textbooks that deal with SCM only, like [10] and [24], cover the traditional way of looking at SCM as consisting of the four activities of: configuration identification, configuration control, configuration status accounting and configuration audit. We have dedicated an entire theme to these formal aspects of SCM. However, in our opinion these aspects may be important, but in the context of a university course there is much more to SCM than that. This means that we treat all four activities in one theme (two lectures and one exercise session). As such we can cover the most important parts, but do not have the time to go into any level of detail. This also has the consequence that we do not find these books suited as textbooks in their entirety as they convey more details than overview. Two books stand out as exceptions to the traditional treatment of SCM as consisting of "the four activities". Babich [6] and later on Mikkelsen and Pherigo [25] take on the developer's perspective and look at SCM as a set of techniques to handle the individual developer's work and his coordination with the rest of the people on a team. Our experience is that students consider these aspects important and we too find that they are central aspects in the practice of SCM. As such they are treated carefully and in detail in themes 2, 3 and 4 – collaboration, workspace and repository respectively. Again neither of these books is sufficient as a stand-alone textbook, as there is more to SCM than just the developer's view – however important it might be.

So far the above references have helped in identifying many topics that should be included in an SCM curriculum. However, none of the references gives any thoughts or advice about how to balance and structure the topics, neither how to prioritise them. If we take all of the above mentioned topics and treat all topics in detail we will have far more material than can be covered even in a course with 50 hours of lectures, exercise and lab sessions. To the best of our knowledge, the only reference that treats the curriculum aspect is the Software Engineering Body of Knowledge (SWEBOK) initiative [27]. It provides an overview of SCM and presents a breakdown of topics along with a description of each topic. It is a very thorough and detailed work that we would very much have liked to follow. However, in our opinion it has the weakness that it considers only traditional SCM, even if it does extend "the four activities" with those of management of the SCM process and release management and delivery. These two extensions are still quite remote from what a developer is usually confronted with. Thus we take note that these six activities are important and should be treated in our course, but do not think that it can serve as the entire curriculum. As an aside, these SCM workshops might not be the best forum for a discussion of the contents of an SCM course as [27] consider these workshops to cover only 5 out of their 44 breakdown topics for SCM.

So we were left with no help when it came down to structuring and prioritising all the topics that we wanted to crowd our course with. In the end we decided to try to come up with what we thought would be a proper curriculum for at university course on SCM. One of the main goals of the course is that the students should master SCM as support for several roles and levels in a company organisation. Fig. 1 depicts four such levels: the individual developer, the project, the company and the society.

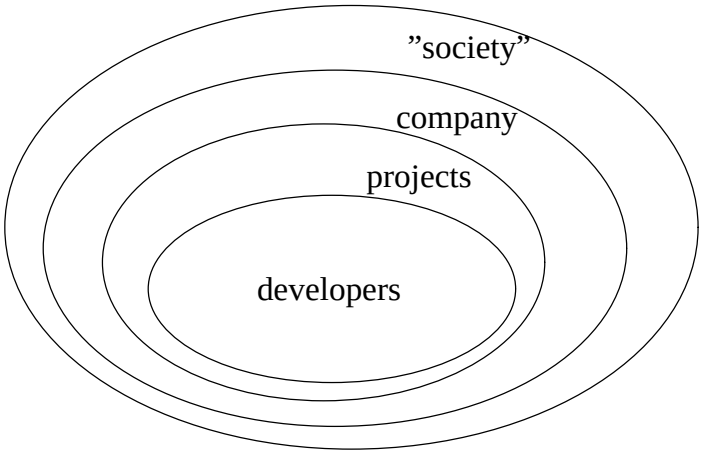

Fig. 1. The different spheres where SCM is practised.

We based the structure and contents on these roles and levels presented in the papers [5] and [9]. The students' career in industry is reflected in the spheres seen from Fig. 1, where they start as developers, have to interact with others in a project, will become SCM-wizards on their project, will have to handle their project's SCM-interaction with the company – and other projects, will become SCM experts that can handle the company's SCM-interaction with customers – and other companies – and will eventually become company-wide SCM experts. This is what we try to reflect in the structure of the course – and in the weight we give each topic/sphere. We spend more time on the innermost sphere and treat that before the others because it is the first that our students will meet when they graduate and get a job. We also do it because we find it easier to build traditional SCM on top on that than the other way around. We also spend less time and detail on traditional SCM as most of our students will probably never have to deal with those aspects directly. However, we find it important that they have some knowledge of all aspects of SCM ranging from what a developer needs to get his daily work done most efficiently to what the company needs to do to provide its customers with consistent products – and why these customers may have SCM requirements to the company. It will also provide them with a proper foundation to build upon if they want to make a career in SCM.

Based on this we can start to structure the course and to prioritise the topics. There should be a progression from developer to company view with main emphasis on the former aspects. Furthermore, for each theme that we define there should be a progression from the basics to more advanced topics. The latter progression we try to obtain

by treating basic concepts in the first lecture and dedicating the second lecture on a theme to a selected set of more advanced topics to be able to treat them in-depth. For the former progression we have theme 3 (workspace) and in part theme 4 (repository) that covers the developer's individual work. The developer's interaction with the project – and as such also with his fellow developers – are covered by theme 2 (collaboration) and by theme 4 (repository), as such describing SCM at the project level with respect to the developers. The SCM-interactions that exist between a project and the company – and between projects – are covered by theme 5 (traditional SCM) – a theme that in reality also treats the SCM-interactions with the customer. Theme 6 (relations to other domains) serve to broaden the students' view of SCM, how it can be practised in different contexts and how other domains can use or inspire SCM. Parts of this theme (PDM) also caters for the needs at the company level

4 Pedagogical considerations

In this section, we discuss the pedagogical considerations that influenced the way we organised and structured the course. These considerations have been sub-ordinate to the contents considerations discussed above. However, we still think that these considerations are important, as they convey important information to understand why we did things the way we did. Furthermore, it can help others to tailor the course if they have other restrictions and traditions for schedule, group work and examination form.

Our first consideration was the prerequisites that the students should have to take the course. In reality there is very little that the students have to know in order to follow a course on SCM – in fact if we take the software out of SCM, many of the concepts and principles would change only marginally. So with regards to knowledge the course does not build on anything in specific and could be taught even at an introductory level. However, for the students to appreciate that there are indeed problems in software development and that a great deal of help can be found in SCM they do need a broad cultural background from software engineering topics and experience from "real" projects. A trend in the curriculum for the undergraduate computer science education in Lund is that the number of student projects increases. One of these is a combined course and project on "Program development in group" [20] in which the students, in groups of 10 during 7 weeks, develop a software product. In this way they learn a lot about practical software engineering. Before the actual project there is a course with lectures and computer lab exercises. One lecture and one lab exercise is about SCM, which is just about enough to help them run the project. It does not give them a deeper knowledge of SCM, however it exposes them to many problems and make them aware that SCM exists and can be helpful. We require that students have taken this project (which is on their second year), because it ensures that they have a varied and broad background in software engineering and because it exposes the students to problems that can be handled by SCM. Being an optional course we thus get interested, motivated and hard working students. We use the same principles of project experience and background for the fair number of exchange students – and graduated students – that want to take the course.

One of the reasons for the prerequisites is based on the philosophy of problem-based learning (PBL) as reflected in the learning cycle of Kolb [22]. It says that we get better and deeper learning if we respect the natural cycle of learning. This cycle starts with experiencing a problem in practice, this is followed by a reflection about what could be the causes of the problem, then we conceptualise a possible solution and finally we plan how to carry it out in practice – at which point we can experience new problems and the learning cycle can continue. During their project course they experienced problems and some students even reflected on the possible causes. Now the course on SCM should help them to conceptualise solutions and plan how they can be carried out. The PBL philosophy reappears in the fact that the students actually try out some of their solutions during the computer labs. It is also the reason for starting at the developer level – where they have practical experience – and progressing through to the company level. By the use of metaphors during the exercise sessions we also try to draw on problems students have experienced that are similar in nature to those caused by lack of SCM. Finally, the structure of each theme into a lecture followed by an exercise session and another lecture and finally a computer lab was also implemented to facilitate PBL.

Dreyfus and Dreyfus [16] explain how people progress from novices through advanced beginners, competent and proficient to become experts. We do not claim – or even hope – to turn our students into experts on SCM. When we get them they are a mixture of novices and advanced beginners in the field of SCM. What we can do is to bring them to the level of competence in SCM – they can only obtain the levels of proficiency and expertise through practising SCM and gaining experience. Something that we can never hope to have time for in a university course. However, we find that this philosophy gives support for the existence of the computer labs and also the fact that exercise sessions are based on discussions and aimed at creating solutions rather than just answers. Furthermore, it gives us a way to explain to the student that they are not experts in SCM even though they have just followed a course dedicated to just that subject. The course have made them competent – and experience will make them become experts in time. This avoids that they start a job interview by claiming that they are experts on SCM.

Bloom [11] introduces a taxonomy for levels of understanding that we can use when we want to specify how far the students come with a given topic. He works with six levels of increasing understanding – knowledge, comprehension, application, analysis, synthesis, and evaluation. Working with such a distinction makes it easier to prioritise each topic. The first three levels are sufficient for users of SCM, while higher levels are necessary for people that have to design SCM. The higher up we get in the spheres of Fig. 1, the less likely it is that our students have to practise – or even design – topics at that level. Therefore, it might be sufficient to know that configuration audit exists even though they do not comprehend it let alone are capable of practising it. The contrary applies for levels close to the developer. Our students should be capable of analysing SCM problems at these levels and to design and maybe even evaluate solutions. Themes 2, 3 and 4 that focus on the developer and project levels are structured such that the first lecture gives the basics and the second lecture gives more in-depth and advanced knowledge. Thus the first lecture brings the students to the level of comprehension or application. The discussions during the exercise ses-

sions contribute to preparing them for the analysis and synthesis levels while the second lecture provides them with analytical tools and a wider and deeper background to synthesise solutions. As such it will enable them to become "SCM wizards" in projects right after the course has finished. For the outer spheres of SCM there is no division in basic and in-depth lectures as we aim for only the first three levels of understanding. Also the SWEBOK [27] project at one point applied Bloom's taxonomy to each of their breakdown topics, clearly aiming at people that had to carry out – as opposed to design – the SCM activities (no breakdown topic exceeds the application level and most topics remain at either knowledge or comprehension, which corresponds pretty much to the levels students obtain in our course).

5 Lessons learned

Reflecting on the experience we have obtained from running this course twice, we want to share some of the lessons that we have learned along the way. What worked and what did not? What are the students' impressions?

The open (metaphor) questions leading to brainstorming like discussions during the exercise sessions works well and is popular among the students. However, it is important to steer them into discussions about more concrete software development problems, i.e. to transfer the metaphor back to the software engineering world (avoid getting stuck in the metaphor). The metaphor is used only in the first phase of the brainstorm – and whenever they get stuck discussing concrete SCM problems/solutions. One way we facilitate the detachment from the metaphor is to insist that they produce a concrete result that they can present for the other groups.

Dividing the course material into themes makes it easy for the students to focus on one aspect at the time and to structure their minds. However, it may also lead to isolated islands of knowledge, which they are not able to combine. Thus, it is very important to actually reserve time to present the larger picture. The overview lecture the first week provides a structure that is then filled out during the rest of the course. The final lecture looks back at the course and makes sure that things are tied together. And finally, the essay that they produce at the last exercise session asks for parts of a "total solution" in order for them to reflect on how all parts work together in practice.

To start with the developer view of SCM works very well. The students are from the beginning well motivated to learn about how to manage situations/problems they have been in themselves. Previous experience from teaching traditional CM, listing the "four activities", is that it bores them to death. However, after three weeks of developer view – bringing them to a level where they are able to analyse different solutions – they are ready to also understand and appreciate the needs and solutions of the manager and the company.

Students gave the computer labs very high ratings in their evaluation of the course. They got a solid understanding of the underlying models for the tools from the mixture of guided exercises and more free experimentation. Even students that had previous self-taught experience with CVS got a deeper insight. Several students wanted labs to be longer than two hours to allow for even more experimentation with the tools. The

first lab on CVS was optional and was aimed at student with no previous experience (not having followed the project that was a prerequisite). We believe the fact that they got a solid understanding of ClearCase in just 2x2 hours of labs to be due to a carryover effect from the CVS labs and we are confident that they would now be able to pick up any CM-tool in 4 hours or less.

Now what about the goals for the course – did we meet them? In our opinion we did not set low standards for passing the course, but still no students were failed. We applied several ways of evaluating the students. During the exercise sessions where the students had to discuss in groups, we were present and circulating between groups taking part in the discussions. This gave us both the possibility to guide the discussions and to make an informal evaluation of the students' level. Furthermore, each group had to shortly present and discuss their findings from the exercises. It is our impression that all students have a very good comprehension of problems and solutions from the developer's perspective. Most students are also very capable of analysing, synthesising and evaluating such solutions. For traditional SCM, students were able to produce fragments of an SCM plan and they were able to evaluate a given SCM plan template with respect to the different roles. We were also present during the lab exercises and could see that students were able to apply and use different work models. We see the fact that they were able to get a solid understanding of ClearCase in just 2x2 hours of lab as an indication of a thorough understanding of the different work models. The final examination was oral and we were able to evaluate in more detail their capability to analyse, synthesise and evaluate SCM problems and solutions. As there were less exercise time for traditional SCM, we were more focused on this aspect during the oral examination. The fact that the examination was done in groups of 3-4 students gave us sufficient time to explore the depth of their learning without sacrificing too much the range of topics.

6 Future work

We have now run two iterations of the course where we have made only minor changes from the first to the second. In general, our experience is that the course is working fine. The students like it and we do not see any big problems that need fixing. However, before running the third iteration we have some ideas for changes that we want to carry out.

What is it that works very well and should be kept? The exercise sessions with the students working in small groups and discussing open questions is definitely a strong point of the course. The computer labs is also something that attract the students and could even be extended as requested by the students. There is room in the schedule for more computer labs and we contemplate utilising this for one or two more labs. We have not decided whether it should be used for adding a third CM-tool, for digging deeper into one of the tools or for exploring completely new aspects like PDM or problem reporting. The structure with themes that fit into one week of activities is also something that we want to maintain. Thus any changes that are made will have to respect that constraint.

The contents of the course seems to be appropriate and has not created any problems. However, this is the point where we see the greatest potential for improving and fine-tuning the course. Spending three weeks on topics from the developer and project spheres and one week on topics from the company sphere may seem a bit unbalanced – and we would not disagree. The reason for the present division is in part due to the schedule constraints mentioned above. However, that does not mean that we could not treat topics that are general for all levels – like the change process – at the developer level and thus only deal with the more formal aspects of already seen solutions at the higher levels. We also contemplate writing our own course material. At the present state we use bits and pieces from many different places. Each part in itself is excellent and to the point. However, the overall result of putting them together is rather like a piece of patchwork.

One thing that we are definitely going to change is to make the students even more active and involved than is the case today. Now all lectures are given by the authors and the students read the literature in parallel. Lectures are not exposing the literature, but are exposing the theme and give a "second opinion" on what the students get from the literature. This could work fine for the basics lectures, but for the advanced topics lectures we plan to have the students do the presentation. 2-4 papers would be selected for each advanced lecture session and each paper assigned to a group. This groups would be responsible for presenting the paper – and all the other groups would be responsible for having questions for discussion. This system would have the consequence that the students would work more actively also with the literature. The groups that had to present a paper on an advanced topic would get an even deeper insight than today, and the rest of the groups would probably also work more with the material during reading. Finally, it would transform the lectures from passive listening to active discussion.

Another thing that we will do in the near future it to cast the course in other contexts. One of the authors has previously given the traditional two lectures on SCM within a general course on software engineering. How could we take the present course and fit it into two or three lectures? Should it be scaled or should we cut it down – and can it be done from our set up or will it have to a course of its own? Furthermore, in the context of a research collaboration with industry, we have to give a one-day course on SCM to future project managers. Again, how can we fit the present 50 hours into one day? Finally, we are considering a post-graduate course on SCM and the above mentioned changes to actively involve students may be one step in turning the present course into a post-graduate course.

7 Conclusions

We have shown that SCM does not have to be confined to a couple of lectures within another course. SCM is rich enough in important topics to fill a whole course on its own. Given more time, the students also reach a higher level of theoretical understanding making it possible for them to analyse and reflect about different situations

and solutions. Students like SCM – out of a student population of 80-100 we have 15-20 students and 4-6 exchange students taking the course

We have made a proposal for the structure and contents of such a course. We have gained some experience giving the course twice and have stated some ideas for changes and improvements. Obviously this is not the final word and we welcome suggestions and discussion that can contrast and complement our experience with other experience from the SCM community.

References

1. B. Appleton, S. P. Berczuk, R. Cabrera, R. Orenstein: *Streamed Lines: Branching Patterns for Parallel Software Development*, http.//www.cmcrossroads.com/bradapp/acme/branching/streamed-lines.html, 1998.
2. U. Asklund: *Configuration Management for Distributed Development – Practice and Needs*, Lund University, 1999.
3. U. Asklund, L. Bendix, H. Christensen, B. Magnusson: *The Unified Extensional Versioning Model*, in proceedings of SCM-9, Toulouse, France, September 5-7, 1999.
4. U. Asklund, I. Crnkovic, A. Hedin, M. Larsson, A. Persson Dahlquist, J. Ranby, D. Svensson: *Product Data Management and Software Configuration Management – Similarities and Differences*, Sveriges Verkstadsindustrier, 2001.
5. U. Asklund, L. Bendix: *A Study of Configuration Management in Open Source Software*, in IEE Proceedings – Software, Vol. 149, No. 1, February 2002.
6. W. A. Babich: *Software Configuration Management: Coordination for Team Productivity*, Addison-Wesley, 1986.
7. M. E. Bays: *Software Release Methodology*, Prentice-Hall, 1999.
8. L. Bendix, P. Nowack: *Software Architecture and Configuration Management*, in 4th Workshop on Object-Oriented Architectural Evolution, Budapest, Hungary, June 18, 2001.
9. L. Bendix, O. Vinter: *Configuration Management from a Developer's Perspective*, in proceedings of the EuroSTAR2001 Conference, Stockholm, Sweden, November 19-23, 2001.
10. H. R. Berlack: *Software Configuration Management*, John Wiley & Sons, 1992.
11. B. S. Bloom (ed.): *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*, Addison-Wesley, 1984.
12. W. J. Brown, H. W. McCormick III, S. W. Thomas: *AntiPatterns and Patterns in Software Configuration Management*, Wiley, 1999.
13. S. B. Compton, G. Connor: *Configuration Management for Software*, Van Nostrand Reinhold, 1994.
14. M. A. Daniels: *Principles of Configuration Management*, Advanced Applications Consultants, 1985.
15. S. Dart: *Configuration Management: The Missing Link in Web Engineering*, Artech House, 2000.
16. S. Dreyfus, H. Dreyfus: *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*, Simon & Schuster, 2000.
17. J. Estublier: *Configuration Management – The Notion and the Tools*, in proceedings of the International Workshop on Software Version and Configuration Control, Grassau, Germany, January 27-29, 1988.
18. J. Estublier: *Software Configuration Management: A Roadmap*, in proceedings of The Future of Software Engineering, Limerick, Ireland, June 4-11, 2000.

19. P. Feiler: *Configuration Management Models in Commercial Environments*, Software Engineering Institute, 1991.
20. G. Hedin, L. Bendix, B. Magnusson: *Introducing Software Engineering by means of Extreme Programming*, in proceedings of the International Conference on Software Engineering, ICSE 2003, Portland, Oregon, May 3-10, 2003.
21. M. Kelly: *Configuration Management – The Changing Image*, McGraw-Hill, 1996.
22. D. A. Kolb: *Experiental Learning: Experience as the Source of Learning and Development*, Prentice-Hall, 1984.
23. D. Leblang: *The CM Challenge: Configuration Management that Works*, in W. F. Tichy (ed.) Configuration Management, John Wiley and Sons, 1994.
24. A. Leon: *A Guide to Software Configuration Management*, Artech House Computer Library, 2000.
25. T. Mikkelsen, S. Pherigo: *Practical Software Configuration Management: The Latenight Developer's Handbook*, Prentice-Hall, 1997.
26. R. S. Pressman: *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 1997.
27. J. A. Scott, D. Nisse: *Software Configuration Management*, in Guide to the Software Engineering Body of Knowledge, Version 1.0, May 2001.
28. I. Sommerville: *Software Engineering, Fifth Edition*, Addison-Wesley, 1995.
29. W. F. Tichy: *Tools for Software Configuration Management*, in proceedings of the International Workshop on Software Version and Configuration Control, Grassau, Germany, January 27-29, 1988.
30. B. White: *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*, Addison-Wesley, 2000.
31. D. Whitgift: *Methods and Tools for Software Configuration Management*, John Wiley and Sons, 1991.