Teaching Software Process with OpenUP

Christian Köppe
University of Amsterdam
Hogeschool Utrecht

June 14, 2010

1 Introduction

Change of an undergraduate curriculum from computer science, often taught as random collections of techniques, to software engineering requires new approaches as both are fundamentally different [1].
The integration of all these techniques into one coherent curriculum is necessary to allow students becoming generalist-specialist software engineers as proposed by Freeman et al. [1].

This work discusses the general possibilities offered by *Software Engineering Processes* (abbreviated as software processes) to reach this goal. Several approaches are known to use different process styles as agile methods or the Rational Unified Process in teaching. One process claiming to integrate these different styles in one process is OpenUP, an open source process framework. This work evaluates OpenUP on its applicability to be used for teaching.

To do this we begin with discussing pedagogical aspects. Then the learning objectives are derived from different sources, namely the definition of software process in SWEBOK [2], the relations between software process and other software engineering disciplines and finally the experiences made with previous students projects at the Hogeschool Utrecht where software process was not integrated.
After that, determination criteria are discussed and formulated which can be used to determine if a process is applicable in an education environment. These criteria are mapped with the defined learning objectives to evaluate if a process fulfilling the criteria can also be used to achieve the learning objectives.
Finally, these criteria are applied to evaluate OpenUP [3] as software process and its possible usage for teaching. It is shown that OpenUP offers the needed characteristics so that it succesfully can be used in teaching.

2 Pedagogical aspects

From a pedogogical viewpoint, there are a couple of best practices to ensure optimal learning by the students. The *Pedagogical Patterns Project* collects and describes patterns based on these practices for education environments [4]. The following patterns can be related to software process:

- *Wider Perspective* [5]: based on ETHOS approach: "to remind engineers that a solution to a technical problem commonly comprises economic (E), technical (T), human (H), organizational (O), and social (S) aspects. These aspects should be integrated into the general structure of a course.".
Especially the human, organizational and social aspects are often under-represented in traditional computer science curricula. All software processes address organizational structures. Furthermore can the trend be observed that also human and social aspects are increasingly integrated in process methodologies, especially agile ones. So a process taking these aspects into account can help with implementing this pattern.

- *Multi Pronged Attack* [5]: choose your examples and exercises so that they cover more than one idea or topic at the same time.

With using a process methodology, it's possible to connect the teaching of a process method with the teaching of specific technical tasks as e.g. defining a software architecture or implementing test cases. That way different topics can be integrated into one project by using a process.

- *Groups Work* [5]: emphasize group work and use both large and small groups and both long-lived and short-lived ones.
This can easily be implemented if using a process. Working in large groups is realized by the team working together for the whole project. Working in small groups is fulfilled by temporary teams realizing specific artifacts, using e.g. pair programming. The process has to support all these forms of group work.

- *Active Student* [6]: keep the students active, in class and out of class.
While the in class activities are not discussed here, implementing the activities out of class are surely of interest. All processes define roles which are fulfilled by the students. In these roles they have to perform tasks (activities) to produce and use artifacts [3]. The required delivery of these artifacts on time and with appropriate quality and quantity doesn't allow students to not participate enough. Therefore a process forces all students in a team to work and being active. Even more, a process guarantees that all students are assigned an equal amount of work (quantitatively and qualitatively), minimizing the effect that only a small part of the group realizes everything.

- *Prefer Writing* [6]: Choose writing exercises over reading exercises.
This can be implemented by requiring the realization and delivery of the artifacts as defined in the process. But it is necessary that the students understand the importance of what they are writing, otherwise it is "busywork that no one ever reads" [7]. If the only purpose of documentation is that it is read (and graded) by the teacher, then it's hard for the students to understand the importance of their work and to place it in the bigger context. Therefore relations/dependencies should exist between artifacts, so that the role of them becomes more clear which helps in increasing the quality.

Another important part is the effective provision of feedback to students so that they are able to improve their work. There are a couple of patterns described in [8] which help in realizing this. Important for a process is that it should offer enough moments where feedback can be provided. Through the definition of phases, deliverables and responsibilities do defined processes offer enough possibilities to provide feedback to students.

Using a process methodology is not only useful from a pedagogical perspective, but also from a technical one. This is examined in the following section and summarized in the learning objectives.

3 The learning objectives of Software Process

To determine the learning objectives of Software Process we first look at the definition as made in the Software Engineering Body of Knowledge (SWEBOK) [2] and the application of this in the Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering [9]. Next the relations between Software Process and the other software engineering disciplines collected in SWEBOK are discussed. Based on this and experiences made in previous student projects at the Hogeschool Utrecht, the learning objectives are formulated at the end of this section.
It also should be considered that there is discussion over how to best approach undergraduate software engineering education: as horizontal and or as vertical specialization [10]. Both approaches are implicitly taken into account in this work.

3.1 The SWEBOK definition of software (engineering) process

Software Process is one of the knowledge areas described in the SWEBOK [2]. As the curriculum is based on SWEBOK, software process is part of it.
The definition is done on different levels: the meta-level and the managerial/technical level. The meta-level comprises the definition, implementation, assessment, measurement management, change, and improvement of the software lifecycle processes [2]. In an undergraduate environment the most important areas are the definition and implementation of a process, which is also reflected in the software engineering education knowledge (SEEK) [9]. The students have to know what the main parts of a general process are (roles, tasks, activities, collaborations etc.) and how these are

applied in a bigger context. Also standard process approaches are discussed.
The curriculum guidelines [9] contain a course on Software Process and Management. This also focuses mainly on the two above described areas, but also contains relations to other disciplines, which is taken into account in the next section.

3.2 Relations between software process and the other disciplines

If examined on the level of technical and managerial activities, software process as described in SWEBOK is related to all other software engineering disciplines [2]. Their artifacts and activities are part of a process definition. People in roles, also defined for processes, perform these activities to use and produce the artifacts.
The importance of the presence and correctness of these artifacts, e.g. software requirements, architecture, test cases or documentation, needs special attention. This can be done if all artifacts and activities are placed in an overall context, typically a process definition. In SEEK this is done by including also parts of software requirements, software quality and management in one software process course [9].

Another view offer Freeman et al. They define five areas as essential elements of software engineering education [1]: Computer Science, Management Science, Communication Skills, Problem Solving and Design Methodology. Most of these areas are related to SWEBOK knowledge areas (KAs) and SEEK knowledge areas. But they explicitly add communication skills and problem solving, which are not addressed directly in SWEBOK and only partly addressed in SEEK [11]. Therefore they offer an interesting viewpoint, where software processes, if addressing both missing areas properly, can add additional value.

3.3 Experiences from previous courses

In the old curriculum at the Hogeschool Utrecht students also had to work on projects. However, often no structure or process was used by the students and required by the teachers. There are some typical characteristics which were observed at these projects:

- Most of the work is done in the last days of the project
- Integration of all individually developed parts is done immediately before delivery/presentation ("big-bang integration"), often leading to not working system parts or surprise behaviour (students common explanation is: "It worked well yesterday!")
- Documentation, incl. requirements documents, architecture etc., is made *after* the implementation and most often not consistent with the code
- Documentation is seen as extra work of no value, whereas source code is seen as the only important artifact
- Planning is made, but not realistic and not followed, testing is mostly planned as last activity close to the deadline, integration is not planned at all
- No assertions on the quality of the product can be made

It was observed that all these characteristics are correlating with bad to average project results, which counts for both the technical results (the delivered solution) and the pedagogical results (improvement of the students skills and knowledge). To prevent the occurence of these characteristics in future projects, learning objectives are formulated in the next section which explicitly address them (objectives 5-7).

3.4 The learning objectives

Taking the three different views as described above into account, we can formulate the learning objectives of Software Process (SP) as follows:

1. The students know standard elements of a software process, e.g. roles, artifacts and activities.

2. The students know the relations between roles, artifacts and activities and their interdependencies.
3. The students can place other software engineering disciplines in the context of a software development life cycle.
4. The students know standard approaches in software processes.
5. The students can do a realistic planning (incl. estimation) of all activities necessary for developing a software system and can follow this plan.
6. The students are able to validate the quality of their delivered artifacts.
7. The students understand the importance/function of their delivered artifacts.

4 Determination criteria for the applicability of a software process methodology

To determine if a software process methodology can be used in an educational environment, two questions have to be discussed:

1. Does it offer the possibility to achieve the learning objectives?
2. Is it applicable for properties and problems specific for an education environment?

The following sections discuss these two questions.

4.1 Learning objectives criteria

Filho discussed in his paper 'Requirements for an Educational Software Development Process' [12] the characteristics needed by a process in order to be applied succesfully in teaching. He describes six required areas, which also this work takes as basic criteria:

1. **Architecture (A)**: Fully defined process steps, Architecture Mapping between process and course milestones, Allowance for iterative work
2. **Team Orientation (TO)**: Adequacy to small teams
3. **Project Cycle Time (PCT)**: Typically 6 months - 1 year time, Possibility of splitting larger projects between courses
4. **Standards and Practices (SaP)**: Exposure to recognized standards and paradigms (UML, CMM, IEEE, ISO etc.), Exposure to widely used industry practices
5. **Student Support (SS)**: Textbooks (including standards), Templates, forms and checklists, Simple and inexpensive tools
6. **Instructor Support (IS)**: Instructor material (slides, examples, exercises etc.)

Some of the environment properties are also tackled with the requirements described by Filho [12]. However, some can be made more concrete and some have to be adapted to fit in the specific environment of the Hogeschool Utrecht. The following list summarizes these:
- team sizes of +/- 4 students
- fixed budget in terms of time available for the students, this is "paid" in the form of earned student credits (ECs)
- fixed deadline (appr. 10 or 20 weeks after begin)

The sufficiency of these requirements for validating if the learning objectives can be reached with a specific process needs to be proven. Therefore we perform a mapping of the above formulated learning objectives and the criteria as described by Filho.

- Objective 1: If *Standards and Practices* are described, a well defined *Architecture* is present and *Student Support* is guaranteed, this objective can be achieved.
- Objective 2: This can also be achieved by *Architecture*.

	A	TO	PCT	SaP	SS	IS
obj. 1	X			X	X	
obj. 2	X					
obj. 3	X					
obj. 4				X		
obj. 5	X	X			X	
obj. 6	X				X	
obj. 7	X					

Table 1: mapping of requirements and learning objectives

- Objective 3: The definition of the *Architecture* has to include necessary activities and artifacts from all related software engineering disciplines, integrated in process steps.
- Objective 4: This can be achieved by *Standards and Practices*.
- Objective 5: To achieve this, the focus has not to be put on process alone, but also on the other disciplines, so that the students based on their knowledge in a specific discipline (e.g. construction or testing) are able to do a realistic estimation. This is implicitly integrated in *Architecture* and as guidelines in *Student Support*. A realistic planning includes the distribution of tasks to team members, which is part of *Team Orientation*.
- Objective 6: Here are the same requirements as for objective 5 besides Team Orientation.
- Objective 7: This can be achieved by a well defined *Architecture*, where the dependencies between the artifacts and the process steps are explicitly considered.

Table 1 gives a graphical summary of this mapping, which helps in determining the most important ones. As can be seen, a good defined process architecture is the main requirement for a software process to achieve the formulated learning objectives. Furthermore, student support and the usage of standards and practices are also important criteria for achieving the learning objectives.
The criteria instructor support and project cycle time do not address learning objectives directly. However, these are also necessary criteria, which assure a suitable teaching environment.

4.2 Additional criteria

From a pedagogical viewpoint, we add a couple of requirements, again based on experiences made in previous students projects at the Hogechool Utrecht:

- The process should not include too many artifacts. It is more important to emphasize the need for a defined process with the minimal necessary artifacts than the amount of artifacts used in the process.
- The artifacts should add value, also in the students perceptions. This doesn't has to be in the beginning of a project (where students typically don't recognize the value), but surely at the end of the project. It is important for the students to recognize why they had done parts of the process, worked on specific artifacts and made a planning.
- The method should also comply to methods used in 'real world' projects, as described in the Real World Experience pattern [6]. Otherwise there is a big chance that the objectives are not reached due to the blocking of the students (the 'why do we have to do this if no one else uses it?'-behaviour).
- The process shouldn't require to be applied completely to assure that the focus can be put on specific disciplines in a project and that the knowledge level of students (corresponding to the place in the study, e.g. year or semester) can be taken into account.
- The process should still be usable if conflicts occur.

- The difference between the knowledge level of all individual team members should be of no impact for application of the process?
- Processes define different roles and it is necessary that the fulfilling of these roles still allows that **all** individual team members can achieve **all** learning objectives (not only the process related ones).
- The validation/acceptance criteria as described in the process should also be applicable for giving feedback to the students or grading, therefore offering 'real world' experience and not artificial one which is only applicable in an education environment.

4.3 Possible problem domains in educational environments

The project domain problems are determined by a few properties:
- the current knowledge level of the students, which responds to the place in the curriculum
- the technical focus of the project, meaning on which of the software engineering disciplines is emphasized, e.g. requirements engineering or software architecture

The SEEK gives a long list of propable system domains [9, 11]. These differ in a wide range and offer diversity which is of sure interest for students, e.g. information systems and dat processing, biomedical systems, multimedia, game and entertainment systems or agent-based systems. Anyway, the choice for which of these can be applied is mainly dependent on the knowledge available by the teachers. Important is that the process method should be applicable for a big part of these domains to offer later flexibility.

5 Known approaches of using software process methodologies in software engineering curricula

Before discussing the applicability of OpenUP we take a look at known approaches of integrating different software process methodologies into software engineering curricula. These can be split in two trends: using the Unified Process (with mostly the Rational Unified Process©(RUP) as specific implementation) and on the other hand using agile methods as eXtreme Programming (XP) or Scrum. The observations made with these are used for discussing the applicability of OpenUP.

5.1 Agile Methods

Agile methods are beginning to find their ways into software engineering curricula. A couple of case studies provide information on how succesful this was [13, 14].

Points mentioned which are relevant for the above described determination criteria :

- If only XP is used in a single software engineering course, the students would lack the skill for documenting and designing larger projects [13].
- XP doesn't require any form of written design documents, but it's essential that students learn design practices as use cases, UML and CRC cards [13].
- Continuous Integration is necessary because difficulties of integration are often underestimated by students (see also the experiences section) [13].
- Working in XP increased overall morale of a students group which was propably due to a working product early in the process [14].
- Arranging meeting times that were acceptable to the pairing partners was a major disadvantage [14].
- XP offered a significant increase in knowledge propagation, reduced overhead in documentation, fewer defects, moderately improved design and noticeable improvement of programming enjoyment [14].
- Students perceptions were that using pair programming resulted in higher level of confidence in code/product and higher quality of code/product compared to traditional software development methodologies [14].

5.2 Unified Process and RUP

There are a couple of case studies about using the Unified Process in software engineering curricula [15, 10]. All these measure the succesful usage of RUP on the amount of implemented functionality in the delivered software. The quality of the other delivered artifacts and their contribution to the overall result is not examined.

Filho evaluated the Unified Process (and RUP) on his appliance to the areas required for an educational software development process [12]. He summarizes that "for educational purposes, it is too heavy and detailed in some aspects, but leaves out a number of software engineering issues that should be addressed in courses.". His main critics are that it seems to be primarily oriented to large teams due to the large number of roles defined ([16] describes 30 roles). There are many artifacts defined (about 69, grouped in 9 sets) but not described formal and the applicability of all of them is questionable.
His conclusion is that a process used for education should be lighter and designed to meet learning objectives.

Bergandy also uses RUP [17], mainly for 2 reasons: as support for project software process and as structure for organizing and scheduling coverage of topics of software engineering. He states that "using RUP provides opportunity for integration and reflection on value of knowledge and skills acquired in previuos courses on a large problem following the whole project lifecycle."

Bergandy's choice for RUP is built on following rationales:

- The RUP framwork is straightforward, well-defined and well documented.
- RUP promotes using Use Cases, students are familiar with UML.
- RUP embraces component-based development as one of the best practices, this concept is already known to the students.
- RUP is becoming the industry standard.
- RUP works with iterations, this offers risk mitigation in students projects, necessary.
- RUP is configurable.

His experiences made were that the RUP framework provides students with a baseline for validation of tasks performed, responsibilities to be fulfilled and artifacts to be produced. The RUP documentation helps teachers to focus on specific parts and helps students [17].

6 Applicability of OpenUP

OpenUP [3] is an open source process framework that is defined as following:

> OpenUP is a lean Unified Process that applies iterative and incremental approaches within a structured lifecycle. OpenUP embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development. It is a tools-agnostic, low-ceremony process that can be extended to address a broad variety of project types.

To determine if OpenUP is applicable we take a look at the core principles and the parts of it. Then we use the above defined criteria to approve the matching of these in order to being applicable in an education environment. As last part also the experiences made with the other approaches are related to OpenUP, which also gives ideas about the applicability.

6.1 Theoretical principles behind OpenUP

OpenUP is an iterative software development process that is minimal, complete, and extensible [3]. It is based on parts of RUP (the four phases and their objectives, the architecture-centric approach,
It defines four mutually supporting core principles:

1. Collaborate to align interests and share understanding
2. Balance competing priorities to maximize stakeholder value
3. Focus on the architecture early to minimize risks and organize development
4. Evolve to continuously obtain feedback and improve

OpenUP knows 7 basic roles (analyst, any role, architect, developer, project manager, stakeholder and tester), which are defined as minimal set of roles. The responsibilities, skills and competencies of these roles are described.

They are 17 work products defined, which are identified as essential artifacts to capture and communicate decisions [3]. These are grouped by five domains: architecture, development, project management, requirments and test. Templates are provided for all work products.

Tasks are defined by domain and focused on results. These tasks are related to roles (as primary and additional performers, also describing how individuals should collaborate to achieve objectives), input artifacts (showing dependencies between tasks and artifacts) and output artifacts.

The OpenUP lifecycles are split in three levels: micro-increments (daily work on work items of individuals), iteration lifecycle (structures how micro-increments are applied, delivers demo-able or shippable build) and project lifecycle (structured into the four phases inception, elaboration, construction and transition).

OpenUP describes practices which are part of the process. These practices are partly corresponding to agile practices, as e.g. described by Beck in [18] and the Agile Manifesto [19].

6.2 What are their claims related to the above described properties and problems?

One of the main statements in [3] is: OpenUP describes the minimal set of roles, tasks (organized by disciplines), and artifacts (organized by work product domains) involved in software development. This implies that it can be applied for all kinds and sizes of software development projects.

They also claim that OpenUP is most useful for four primary groups of users:

- **Software Development Practicioners (working together as a project team)**: The practicioners (in this case the students) find guidance on the requirements related to the defined roles, specifically in the activities, artifacts and collaborations of roles.
- **Stakeholders**: The stakeholder group is of less interest here (but from the practicioners view the role of stakeholder is important and mostly fulfilled by a lecturer/teacher).
- **Software Process Engineers**: Software process engineers are able to extend and modify OpenUP. This could be of interest if it shows that OpenUP as delivered is not completely suitable for reaching the learning objectives described above.
- **Instructors**: One of their claims is that OpenUP is also appropriate for academic organizations. It can be used as basis for software engineering courses or courses in software process engineering.

According to these claims OpenUP should be applicable for use in an education environment. To discuss this we determine if OpenUP offers all properties to fulfill the requirements described earlier in that work.

6.3 Comparing requirements and OpenUP properties

The first part of this section compares the requirements stated above with the properties. Then we also look at the experiences made with agile methods and the Unified Process in educational environments. Some of their principles and process parts can also be found in OpenUP, so that they are comparable.
Finally we examine what OpenUP says about using it under certain properties and for certain problems and see if this fits.

6.3.1 Core Requirements

Architecture
OpenUP offers fully defined process steps. All tasks, their ordering and their dependencies are described and related to roles and artifacts.
OpenUP knows different lifecycle layers: micro-increments, iteration lifecycle, and project lifecycle. The project lifecycle is split in four phases where each phase defines a milestone as acceptance criterium. These different lifecycle layers allow easy mapping with course milestones on different levels, offering a high flexibility.
Working in iterations is one of the management practices of OpenUP.

Team Orientation
OpenUP defines a couple of roles, which are all interrelated and important for reaching the goals of developing a software system. This can only be done if working in a team. Furthermore, OpenUP includes the practice *Whole Team*, also known from XP [18].

Project Cycle Time
As described above, OpenUP knows different lifecycle layers. These can be easily adapted to fit the typical project durations in education. The only adjustment which propably has to be made is the amount and length of the iterations used per phase.

Standards and Practices
OpenUP uses UML as standard notation. It includes practices as defined in eXtreme Programming [18] and other agile methods. Furthermore it provides checklists, templates and guidelines, which also can be used for validation and quality improvement.

Student Support
OpenUP is released as web-based description and can be downloaded for free [3]. It includes descriptions of all parts of the process and uses hyperlinks to support the exploration of it. All relations and dependencies are visually shown, making it for students easier to explore and learn than conventional text books.
OpenUP includes templates for all artifacts, helping students with the working on these artifacts. It also includes extensive guidelines on all practices.
OpenUP furthermore describes basic parts of the process to some detail. It therefore offers good student support.

Instructor Support
Actually, everything which is available for students is also available for instructors, but nothing more. If instructors want to integrate more basic concepts in a course or other software process related topics, this has to be developed independent of OpenUP. OpenUP also offers no help in assessing the quality of the delivered artifacts and the fulfilling of the roles. Therefore, instructor support is not good fulfilled.

6.3.2 Additional Requirements

Not too many artifacts
OpenUP claims that it contains only the minimal set of necessary artifacts needed in a software development project.

artifacts should add value

Same as above. Furthermore the relations of artifacts and tasks are explicitly described in OpenUP, all artifacts are required inputs for some tasks and therefore valuable.

real world experience
OpenUP is already used in 'real world' projects [20]. It is also based on well-known agile practices and the structure of RUP, which both are widely used.

partly appliance of a process
OpenUPas process framework can be customized. However, because it describes a minimal process, no parts should be left out. An alternative is to provide students with artifacts which they can use or to let them fulfill not all roles. That way it is still possible to put the focus on technical parts andd still use OpenUP.

still usable if conflicts occur
Actually, OpenUP doesn't say anything about how to take care of conflicts between team members. So nothing can be said about this. But OpenUP tries to prevent conflicts by using the Whole Team practice.

able to handle different knowledge levels in a team / different roles still allow all team members to achieve the learning objectives
OpenUP defines 7 roles and tasks which are performed by them. but for each task not only a primarily performing role is defined, but also additional roles which help. It has to be taken care of (by team members and by teachers) that also the additional roles are fulfilled in the tasks. This way, even if a student only has one role, she still can perform a lot of tasks, which increases her knowledge level.

'real world' acceptance criteria
OpenUP provides guidelines and templates which are also used in the 'real world'. By doing this, the same criteria are used to determine if an artifact meets the quality criteria. That way it is assured that the knowledge of how to produce these artifacts can also be applied in real commercial projects.

6.3.3 Pedagogical Patterns

Wider Perspective
The technical and organizational aspects are covered in the domains and the tasks/artifacts/roles. OpenUP includes the Whole Team practice, which is addressing partly the human and social aspects of software development. This surely needs more attention, especially because sometimes interpersonal problems between team-members can arise annd have to be handled.
It explicitly doesn't address the economic aspect, so this has to be added if necessary.
Actually, most of the aspects are integrated in OpenUP, therefore a wider perspective can be achieved.

Multi Pronged Attack
OpenUP covers a few of the most important disciplines of software engineering by organizing all roles, tasks and artifacts in 5 domains. These are architecture, development, project mmanagement, requirements and test. Therefore it is possible to integrate these all in one project, offering the required multi pronged attack.

Groups Work
See also the Team Orientation requirement. Furthermore offers OpenUP the possibility to let students work in groups of different granularities: The whole team for the whole project and small temporary teams for the realization of work products.

Active Student
All roles are needed in a project and their fulfillment (incl. delivery of the artifacts) is essential for a succesful project. This means that all students have to fulfill their roles according to the planning. Furthermore, the dependencies between artifacts and tasks ensure that all students are active over the whole project lifecycle, because of the regular moments of delivery.

OpenUP offers a good basis to keep students active.

Prefer Writing
All artifacts have to be written down or modeled in some other way. All roles have artifacts associated with them, so the students fulfilling these roles are also responsible for the delivery of the artifacts. This way it is ensured that the students not only do, but also document what they are supposed to do or have done.

Real World Experience
OpenUP is also used in real world projects, e.g. as standard methodology at the Klaverblad Insurance Company [20] and can therefore be used to implement the Real World Experience pattern.

6.3.4 Comparable Experiences - Agile Methods

Using only XP leads to lacking of documentation and design skills
Other than with XP, OpenUP includes more documentation and design documents (based on design practices as UML and use cases) so that these skills can be achieved by using OpenUP.

Continuous Integration is necessary because difficulties of integration are often underestimated by students
OpenUP also includes Continuous Integration as practice, therefore this part has been paid attention.

Working in XP increased overall morale of a students group which was probably due to a working product early in the process
This is also offered by OpenUP through working in iterations, continuous integration and the delivery of an executable architecture prototype in the early phase of the project.

Arranging meeting times that were acceptable to the pairing partners was major disadvantage
OpenUP does not enforce pair programming, so this is no problem.

XP offered a significant increase in knowledge propagation, reduced overhead in documentation, fewer defects, moderately improved design and noticeable improvement of programming enjoyment
While OpenUP surely offers good ways of knowledge propagation, it can't be said if it really leads to fewer defects, improved design or noticeable more programming enjoyment. This has to be evaluated using data from students projects.

Students perceptions were that using pair programming resulted in higher level of confidence in code/product and higher quality of code/product compared to traditional software development methodologies
While OpenUP doesn't explicitly includes pair programming, this practice can easily be added to it (as OpenUP is a process framework and can be adjusted). On the other hand are these experiences based on the *perceptions* of the students and not the *actual* quality of the delivered products.

6.3.5 Comparable Experiences - (Rational) Unified Process

Bergandy's arguments for using RUP were:

- The RUP framwork is straightforward, well-defined and well documented.
- RUP promotes using Use Cases, students are familiar with UML.
- RUP embraces component-based development as one of the best practices, this concept is already known to the students.
- RUP is becoming the industry standard.
- RUP works with iterations, this offers risk mitigation in students projects, necessary.
- RUP is configurable.

Besides becoming an industry standard (which is also questionable for RUP) and being component-based, OpenUP offers all arguments used by Bergandy.
The architecture-centric approach of OpenUP also includes the usage of components, the same goes for the appliance of UML. Therefore OpenUP can be used for component-based development. OpenUP is already used in some companies as standard process [20].

7 Conclusion

Software Processes play an important part in a software engineering curriculum. They are not only one of the disciplines, but also offer the possibility to integrate other disciplines in projects. This way not only a couple of pedagogical best practices can be realised, but also a broader specialisation in the software engineering field.

Three different views on how to define the learning objectives are discussed. First, pedagogical patterns were mined for their probable relation with software process. Then the SWEBOK definition of software process and the integration of software process as course in a software engineering curriculum as described in SEEK were discussed. Based on these examinations, seven learning objectives were formulated.
A known approach of determining criteria for an educational software process is presented and expanded with criteria based on experiences made with students projects at the Hogeschool Utrecht. These criteria are mapped with the defined learning objectives to show that these criteria can also be used in this work.
Finally these criteria were used to evaluate the applicability of OpenUP as educational software process.

It has been shown that OpenUP is applicable as process to be used in undergraduate education environments. It offers nearly all properties needed to fulfill the requirements determined for the applicability of software processes in teaching.

8 Future work

The conclusions made can be approved or rejected by a case study with students using OpenUP.

Another important part which is often missing is a lack of visibility of what is actually learned [13]. This also plays a role in applying OpenUP (or another process) in students projects. The known case studies ([14, 13, 17, 15]) mostly assess only the perceptions of the students and not the actual results. Future work could focus on the real delivered quality, e.g. by using metrics etc.

References

[1] Peter Freeman, Anthony I. Wasserman, and Richard E. Fairley. Essential elements of software engineering education. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 116–122, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[2] IEEE Computer Society. *Software Engineering Body of Knowledge (SWEBOK)*. Angela Burgess, EUA, 2004.

[3] Openup versie 1.5.0.4. `http://www.eclipse.org/epf/downloads/openup/openup_downloads.php`, 2010. Retrieved June 10, 2010.

[4] Pedagogical patterns project. `http://www.pedagogicalpatterns.org/`, 2010. Retrieved June 08, 2010.

[5] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, Helen Sharp, and Marianna Sipos. Teaching from different perspectives. `http://www.pedagogicalpatterns.org/`. Retrieved June 06, 2010.

[6] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, and Helen Sharp. Patterns for active learning. `http://www.pedagogicalpatterns.org/`. Retrieved June 06, 2010.

[7] Mohamed E. Fayad. Software development process: a necessary evil. *Commun. ACM*, 40(9):101–103, 1997.

[8] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, and Helen Sharp. Feedback patterns. `http://www.pedagogicalpatterns.org/`. Retrieved June 06, 2010.

[9] Joint Task Force on Computing Curricula. Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering. Technical report, IEEE CS and ACM, 2004.

[10] M. Brian Blake and Todd Cornett. Teaching an object-oriented software development lifecycle in undergraduate software engineering education. Department of Computer Science Georgetown University.

[11] Timothy C. Lethbridge, Richard J. LeBlanc Jr, Ann E. Kelley Sobel, Thomas B. Hilburn, and Jorge L. Diaz-Herrera. Se2004: Recommendations for undergraduate software engineering curricula. *IEEE Softw.*, 23(6):19–25, 2006.

[12] Wilson P. Paula Filho. Requirements for an educational software development process. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 65–68, New York, NY, USA, 2001. ACM.

[13] L. Williams and R. Upchurch. Extreme programming for software engineering education? In *FIE '01: Proceedings of the Frontiers in Education Conference, 2001. on 31st Annual*, pages T2D–12–17vol.1, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Linda B. Sherrell and Jeff J. Robertson. Pair programming and agile software development: experiences in a college setting. *J. Comput. Small Coll.*, 22(2):145–153, 2006.

[15] Jiang Li. Teaching unified process in software design and development courses: a case study. *J. Comput. Small Coll.*, 24(5):5–11, 2009.

[16] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[17] Jan Bergandy. Teaching software engineering with rational unified process®(rup). In *ASEE 2006 Annual Conference*. ASEE, 2006.

[18] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[19] The agile manifesto. `http://www.agilemanifesto.org/`, 2010. Retrieved June 05, 2010.

[20] Peter van Ppoppel, Ludi Cosman, and Ad Strack van Schijndel. Openup in de praktijk. *Software Release Magazine*, 14(2):26–29, april 2009.