Mining Student Capstone Projects with FRASR and ProM

Wouter Poncin  Alexander Serebrenik  Mark van den Brand
Eindhoven University of Technology, Eindhoven, The Netherlands
w.poncin@alumnus.tue.nl, {a.serebrenik, m.g.j.v.d.brand}@tue.nl

Abstract
Capstone projects are commonly carried out at the end of an undergraduate program of study in software engineering or computer science. While traditionally such projects solely focussed on the software product to be developed, in more recent work importance of the development process has been stressed. Currently process quality assessment techniques are limited to review of intermediary artifacts, self- and peer evaluations. We advocate augmenting the assessment by mining software repositories used by the students during the development. We present the assessment methodology and illustrate it by applying to a number of software engineering capstone projects.

Categories and Subject Descriptors K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education; H.2.8 [Database Management]: Database applications—data mining

General Terms Management, Measurement

Keywords capstone project, mining software repositories, software engineering

1. Introduction
Undergraduate computer science or software engineering curriculum is often crowned by a capstone project, providing the students with the opportunity to undertake a significant software engineering project on their own. Recommended by such model curricula as [3, 4], capstone projects build upon the knowledge gained from courses throughout the program, and should includes development of requirements, design, implementation, and quality assurance [3]. While traditionally such projects solely focussed on the software product to be developed, in more recent work importance of the development process has been stressed [13, 30].

Growing importance of the development process poses new challenges to instructors acting as assessors: quality of the software development process has to be assessed and taken into account while determining the students’ grades. The importance of process quality assessment has been recognized, e.g., in [18]. In practice to obtain insights in the development process the assessors resort to evaluation of artifacts produced during the development (e.g., project management plans or requirements specifications), possibly augmented with the self- and peer assessment [5, 19].

To improve on the objectivity of the assessment as well as to provide the learners with prompt feedback, we propose to augment the existing assessment techniques with software repository mining, i.e., analysis of the development process based on repository data. Software repositories such as source-code management systems, document repositories and issue-tracking systems, are commonly seen as a part of the project infrastructure [31], and, hence, are readily available for the analysis. A plentitude of data available in such repositories triggered an extensive research effort on repository mining [6, 12, 14, 24]. Specifically, we suggest applying process mining to analysis of data from multiple software repositories. Process mining [10] has been developed to extract information from event logs produced by an information system, and since then has been demonstrated to be a valuable technique for analyzing business processes in various domains [2, 23]. Moreover, recently it has been successfully applied to analyze and visualize information from multiple software repositories, including version control systems, issue-trackers and mail archives [21].

The main advantage of process mining applied to software repositories is a clear separation between the preprocessing step and the analysis step. The preprocessing step extracts information from software repositories and combines information into an event log, while the analysis step aims at discovering the process structure, reflected in the log, and analyses its correctness or visualizes it. Both steps are supported by tools: FRASR [21] and ProM [10], respectively. Both tools are highly flexible allowing the instructor to answer a multitude of questions pertaining to the software development process, followed by the students.

The remainder of the paper is organized as follows. We start with a brief discussion of the assessment methodology...
proposed and the supporting tools in Section 2. Next in Section 3 we introduce the capstone software engineering project as organized at Eindhoven University of Technology, identify the instructor’s questions pertaining to the process assessment and show how these questions can be answered using the methodology proposed. Related work is discussed in Section 4 and, finally, Section 5 concludes the paper.

2. Assessment methodology

Development process quality assessment begins with identifying a process-related assessment question. This step is carried out by the instructor depending on the project learning goals associated with the development process. Next, using FRASR, the instructor combines information from different software repositories into one event log, imports the log in ProM and analyzes it using ProM mining and visualization plugins to answer the initial assessment question.

2.1 Event logs

In order to be amenable for process mining, the event logs should conform to the process mining event log meta model\(^1\) shown in Figure 1. An event log contains data from a number of processes (usually one). Each process has a number of process instances, also known as case instances, that can be uniquely identified. Furthermore, each process has a number of activities, and each process instance—a number of events, consisting of an activity being executed at a certain moment in time and associated with certain data.

For instance, a log of an insurance company might contain information about a billing process and a refund process. A refund process has a number of process instances uniquely identified by the claim number. Activities that should be executed in the refund process may include registering the claim, and checking the insurance policy. An example of an event is “On Thursday September 23, 2010 Alice checks the insurance policy of the persons involved in claim 478-12”. Process mining aims, therefore, at discovering the information about a process, based on the information about different process instances. The preprocessing step produces a log conforming to the process mining event log meta model. Given such a log, process mining techniques can, for instance, derive abstract representations of the process control flow or detect relations between the individuals involved in the business process and their tasks.

2.2 Preprocessing with FRASR

The preprocessing step commences with defining data sources, i.e., providing FRASR with information about software repositories. Data sources can be defined either as a local file or by entering the repository URL and authentication information.

Next, the user has to choose the way related events should be associated with each other by means of a case mapping and then has to define ways to extract the events from the data sources. For instance, one can consider student as a process instance, where the associated events are, e.g., mails sent by the student or file modifications performed by her as recorded in the version control system. Alternatively, one can consider a software component as a process instance, where the associated events are, e.g., bugs or code modifications associated with this component.

Furthermore, FRASR supports developer matching definition: indeed, the same student might be using different or multiple user names in different software repositories. Developer matching can be either defined manually by the instructor, or determined using heuristics.

Finally, the combined event log can be exported either as a comma-separated list or in the MXML format, supported by ProM. More information about FRASR can be found in [9].

2.3 Mining with ProM

As opposed to FRASR, ProM does not enforce a well-defined sequence of processing steps. However, it provides the user with more than 170 mining, analysis, monitoring and conversion plugins.

Mining plugins aim at discovering a process model for a given log: e.g., the Alpha algorithm extracts a control flow. Analysis plugins aim at providing insights in correctness or performance of the process reflected in the log. For instance, Dotted Chart Analysis [26] shows a spread of events of an event log over time. Conversion plugins are responsible for transformations between different process model formats, such as BPMN, eEPC and Petri nets, while the monitoring plugins observe the process behavior, detect deviations with respect to the expected behavior, try to diagnose the origin of the problem and to resolve it.

More information about ProM can be found in [10].

---

\(^1\) As process mining terminology was evolving, different papers use different names for notions as process, event, etc. In our choice of names in Figure 1, we follow [21].
3. **Capstone project at Eindhoven University of Technology**

Capstone software engineering projects have been introduced at Eindhoven University of Technology in 1997/98. Since then hundreds of students have participated in these projects, including the first author. The project aims at familiarization of the students with working in a large non-trivial software project. The capstone projects are carried out by groups of 7–10 third-year undergraduate computer science students developing a software system for real customers. Cooperation with a real customer has been reported to provide the students with an opportunity to increase their occupational identity [17]. Customers involved in capstone project at Eindhoven University of Technology include SMEs and multinationals, research institutions and nonprofit organizations.

During the project the students are assisted by a project manager (a master student), a technical advisor (a member of the academic staff) and the senior management (senior academic staff members). Students are expected to develop a software system following the well-known V software development model and adhering to the guidelines of the European Space Agency [1], i.e., to produce a number of prescribed management documents, such as Software Quality Assurance Plan (SQAP), a number of prescribed product documents, such as User Requirements Document (URD), a prototype and finally an implementation. The implementation should be delivered to the customer and accepted/rejected by her based on the original requirements and the corresponding acceptance test plan (ATP).

Every two weeks the project manager and one of the students have an appointment with the senior manager to discuss the project progress, e.g., timeliness of the delivered documents, task distribution balance within the group, adherence of the process followed to the ESA standard and to the V-model. A technical advisor is present during weekly group meetings and provides the group with feedback on the technical quality of the product delivered as well as on adherence of the intermediate product documents to the ESA standard. Should the students fail to meet the expectations of the senior management or of a technical advisor they can decide to intervene and implement a corrective action, e.g., by reassigning the students to different tasks within the group.

Upon the project completion approximately fifteen aspects of the project work are evaluated: the customer evaluates her degree of satisfaction, the technical advisor—quality of the product and of the intermediate product documents, while the senior management assesses the process-related aspects of the project mentioned above. The final grading is done by the senior management.

So far, the only means to verify, e.g., adherence to the ESA standard, required both for the intermediate feedback and for the final assessment, were interviews with the students, project managers and technical advisors augmented by scrutiny of development artifacts submitted by the students. Based on the interviews with the senior management, we have identified three questions to be addressed in this study: prohibition of the prototype reuse, exemplifying adherence of the process followed to the ESA standard, work distribution among the students and adherence to the V-model.

The ESA standard prescribes development of a prototype at the software requirements stage. While prototyping is beneficial when high risk functional, performance or user interface requirements are concerned, it commonly represents only a limited view of the system and often ignores quality, reliability, maintainability and safety requirements. Therefore, the ESA standard explicitly prohibits delivering the prototype as part of an operational system [1]. Hence, the first process-related software engineering question we consider, is whether the prototype was reused as a part of the final implementation delivered to the customer.

While software development in the industrial setting usually involves specialized project members, e.g., architects, developers, testers and technical writers, this is not necessarily the case in capstone projects. In fact, one could argue that one of the learning objectives could be that all students should demonstrate their ability to play all aforementioned roles. At Eindhoven University of Technology the decision how each one of the students should contribute to the project is left to the students themselves. Senior management is still, however, interested in obtaining insights in the work distribution within the group.

Finally, the students are required to follow a slightly modified version of the traditional V model. While in the traditional V model the subsequent software development phase can commence only once the preceding phase has been successfully finished, the modification used for the capstone projects recommends starting the subsequent phase a little bit before the deadline of the preceding phase. Senior management motivates this modification by the need to take the students’ learning process into account: while experienced engineers are on beforehand aware of the constraints the subsequent phase of the V model imposes on the deliverables of the preceding phase, the students have yet to discover these constraints. If constraints are discovered early enough, i.e., before the completion of the preceding phase, the deliverables of the preceding phase can be easily adapted to meet these constraints. If constraints are, however, discovered once the preceding phase has been completed, any modification of the deliverables would require implementing a change request procedure prescribed by the ESA standard.

Our decision to assess the development process based on software repositories relies on the assumption that all information required for such an assessment is present in the repositories. Validity of the following conclusions can be threatened if this assumption is violated. To ensure the independence of the analysis, it has been carried out by the
<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>#</th>
<th>Customer</th>
<th>Subversion</th>
<th>Additional data sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>2005</td>
<td>8</td>
<td>Multinational</td>
<td>2082</td>
<td>none</td>
</tr>
<tr>
<td>II</td>
<td>2006</td>
<td>9</td>
<td>SME</td>
<td>1069</td>
<td>none</td>
</tr>
<tr>
<td>III</td>
<td>2007</td>
<td>7</td>
<td>Biomedical research institute</td>
<td>1134</td>
<td>none</td>
</tr>
<tr>
<td>IV</td>
<td>2008</td>
<td>10</td>
<td>Multinational</td>
<td>2032</td>
<td>234 Trac tickets with 505 modifications/comments, 71 Trac Wiki articles with 178 modifications, 124 mail messages organized in 73 threads</td>
</tr>
<tr>
<td>V</td>
<td>2009</td>
<td>7</td>
<td>Nonprofit organization</td>
<td>1574</td>
<td>20 Trac tickets with 51 modifications/comments, 66 Trac Wiki articles with 204 modifications, 204 mail messages organized in 130 threads</td>
</tr>
<tr>
<td>VI</td>
<td>2009</td>
<td>8</td>
<td>SME</td>
<td>1553</td>
<td>409 mail messages organized in 218 threads</td>
</tr>
</tbody>
</table>

Table 1. Capstone projects considered

first author supervised by the second author, while detailed information about the projects was available to the third author, who acted as an expert validating the results obtained.

3.1 Projects considered

Table 1 summarizes data of the six capstone projects we consider in our study. For each project we indicate the year when it has been carried out, the number of students involved not including the master student acting as the project manager, type of the customer, number of revisions in the Subversion repository (all projects made use of Subversion repositories) and additional data sources available.

In the remainder of this section we study prototype reuse, developers’ roles within the group and adherence to the V-model. For the first study, FRASR had to export data from the software repositories and to create the event log. On a 32bit Windows 7 machine with an Intel Core2 Quad CPU @ 2.40 GHz with 3GB of memory this took between 10 seconds and 4 minutes for the various capstone projects. Since FRASR uses caching, the subsequent studies no longer required exporting data from the repositories, i.e., the existing data had only to be rearranged to create a new log. Time required to rearrange the log was negligibly small.

3.2 Prototype reuse

The ESA standard explicitly prohibits reusing prototype software as a part of the system implementation delivered to the customer: “Prototypes usually implement high risk functional, performance or user interface requirements and usually ignore quality, reliability, maintainability and safety requirements. Prototype software is therefore ‘pre-operational’ and should never be delivered as part of an operational system.” [1]. In order to assess whether the students have followed the ESA prohibition to reuse the prototype as a part of the final implementation, we have considered the information present in the version control system.

Each capstone project we have studied had one Subversion repository that contained all files produced for the project. The files contained in the Subversion repository are used as process instances, while the actions performed on them (addition, modification, deletion and renaming) are used as the events. The developer matching has been constructed manually based on the Software Project Management Plan. The resulting event log has been exported in the MXML format. Next we have imported the log to ProM and using ProM we have filtered out all non-code related files, such as process or product documents. Figure 2 shows the Dotted Chart visualization obtained by ProM for the Project IV data. The files are displayed vertically, sorted by creation time, and colored by ‘modification type’ (additions are colored blue, modifications—white, deletions—red, and renamings—black). Two distinct triangles can be identified.

Each of those triangles indicates a period in which related files have been modified (added, deleted or renamed). The left triangle ranges from the beginning of the project until halfway the project and the second triangle ranges from halfway the project until the end of the project. As the events in the left triangle end close to the deadline of the software requirements phase, there is an indication that these files belong to the prototype, which should have been developed during this phase [1]. Furthermore, as the prototype files were not modified after the deadline, we see that the prototype files have not been reused during the actual development. It should be noted that one could argue that the final implementation could have been obtained by copying the prototype implementation files to a new folder. This is, however, also not the case in the project on Figure 2: indeed massive file copying would have resulted in a large vertical blue line (similar to the line around 2008-05-20, corresponding to the creation of a development branch).

In four out of five further capstone projects we have analyzed (Table 1), a similar pattern of two triangles has been observed, i.e., these projects conform to the ESA guideline that the prototype may not be reused in the final implementation. These observations were confirmed by the interviews with the group members and project managers. In Project III, however, only one triangle was observed (Figure 3). Presence of one triangle could be interpreted as absence of the prototype, as prototype not being stored in the version control system or as prototype evolving to the final implementa-
Figure 2. Project IV: the triangle on the top corresponds to the prototype, the triangle on the bottom corresponds to the final implementation.

Since the prototype is supposed to be developed during the software requirements phase, absence of a prototype or prototype being stored outside of the version control system would mean that the triangle should start not earlier than at the architectural design phase. This, however, turned out not to be the case for the student project as the triangle started at the beginning of the project. Hence, in this project the students reused the prototype implementation instead of implementing the functionality from scratch.

Figure 3 suggests, therefore, that should FRASR+ProM have been used during the run of Project III, the senior management or a technical advisor could have intervened in the development process and suggested the students to reconsider the development strategy.

3.3 Developers’ roles

The second question we address pertains to the task distribution balance within a group. Accurate individual assessment of group project participants is a well-known challenge [8, 22]. In this section we study how one can get a better insight in contributions of individual students using FRASR+ProM. As the case study we have chosen the capstone project with the largest amount of data available, i.e., Project IV. Data sources available for Project IV included the Subversion version control system with 2032 revisions, 234 Trac tickets with 505 modifications/comments, 71 Trac Wiki articles with 178 modifications, and 124 mail messages organized in 73 threads. This project involved eleven students: ten bachelor students acting as software developers and one master student acting as the project manager. While we could have eliminated the project manager a priori, we have opted to keep this information as a form of a sanity check. Project manager can be expected to have a very different contribution to the project than the other students, and this role should have been easily identifiable.

In order to check how the work has been developed between the project participants, we have included information from all data sources available. We have categorized all the artifacts such as repository files and mails as being related to one of the documents prescribed by ESA, to the process organization (e.g., meeting minutes), or to the implementation. We no longer distinguish between the prototype and the final implementation. We have also created a separate category “null” for a small number of events that cannot be classified according to the aforementioned categories. From the FRASR perspective the categories are used as process instances, while the actions performed on them (e.g., sending an e-mail) are used as the events. As above, the developer matching has been constructed manually based on the Software Project Management Plan. The resulting event log has been exported in the MXML format and imported to ProM.

For each category and each project member we determine the number of events contributed by the project member to the category. This step has been accomplished by the so called “originator by task” matrix, calculated by ProM and shown in Table 2. Next, for each category we calculate the share of events associated with the specific project member. Based on these shares we have calculated the cosine similarity [25] between the project members, and finally, for each project member we have determined the average similarity with other project members.

The project member most dissimilar to other project members is D (average cosine similarity with other project members is $\sim 0.0369$). Indeed, the contribution of D is limited to agendas, minutes and Software Project Management Plan. One can conjecture that D was a project manager. This conjecture has indeed been confirmed by the Software Project Management Plan.

The second most dissimilar project member is F (average cosine similarity measure $\sim 0.0858$). While F contributed to the code, he was the one with the highest share of events pertaining to the Software Verification and Validation Plan and to test plans (ATP, ITP, STP and UTP), i.e., F can be labeled as the “dedicated tester” of the project. We also see that the
Table 2. Project IV: activities of different project members. Higher level categories such as “Management documents”, “Testing plans” and “Process execution” have been added manually.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12 2 0 0</td>
<td>5 18</td>
<td>52</td>
<td>0</td>
<td>90</td>
<td>1 0 0 4</td>
<td>1 1</td>
</tr>
<tr>
<td>B</td>
<td>6 11 4 3</td>
<td>25 51</td>
<td>25</td>
<td>2</td>
<td>184</td>
<td>1 0 1 11</td>
<td>5 6</td>
</tr>
<tr>
<td>C</td>
<td>0 0 0 0</td>
<td>9 0</td>
<td>65</td>
<td>1</td>
<td>57</td>
<td>0 0 0 0</td>
<td>0 7</td>
</tr>
<tr>
<td>D</td>
<td>5 0 0 0</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
<td>12</td>
</tr>
<tr>
<td>E</td>
<td>0 0 0 0</td>
<td>0 0</td>
<td>1</td>
<td>0</td>
<td>82</td>
<td>0 0 0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>F</td>
<td>0 0 0 14</td>
<td>1 7</td>
<td>2</td>
<td>0</td>
<td>65</td>
<td>0 0 0 31</td>
<td>6 1</td>
</tr>
<tr>
<td>G</td>
<td>0 0 0 0</td>
<td>10 28</td>
<td>8</td>
<td>5</td>
<td>108</td>
<td>0 0 0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>H</td>
<td>0 0 0 0</td>
<td>1 1</td>
<td>31</td>
<td>6</td>
<td>285</td>
<td>0 0 0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>I</td>
<td>0 0 16</td>
<td>8 30</td>
<td>65</td>
<td>2</td>
<td>97</td>
<td>0 0 0 0</td>
<td>0 9</td>
</tr>
<tr>
<td>J</td>
<td>1 23</td>
<td>0 0</td>
<td>11</td>
<td>33</td>
<td>52</td>
<td>0 0 0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>K</td>
<td>0 0 0 0</td>
<td>1 56</td>
<td>65</td>
<td>6</td>
<td>84</td>
<td>0 0 0 0</td>
<td>1 22</td>
</tr>
</tbody>
</table>

developer with the most significant contribution to the implementation is H, who according to the project documents was also responsible for the implementation effort. Furthermore, we see that while all the developers were involved in the architecture design (ADD), the software requirements phase shows a clear distinction between those that have been actively involved (e.g., B and J) and those that have not been (e.g., E and H).

Since one of the aforementioned learning objectives is that all students should demonstrate their ability to play all roles, students focussing on one development role can be considered to be undesirable. As in case of the prototype reuse in Project III (Section 3.2) we observe that should FRASR+ProM have been available during the run of Project IV, the senior management could have considered to intervene and to suggest reassigning student developers to different tasks, e.g., reassigning F to a non-testing task.

3.4 Development model

The prescribed development model for the software engineering projects, is the V development model, in which the phases are executed sequentially. Furthermore, a test plan has to be created for each subsequent phase. As explained above, the capstone projects at Eindhoven University of Technology follow a slightly modified variant of the V model, allowing a limited overlap between two adjacent phases.

To study the adherence to the modified V model we consider management documents and product documents as process instances. We also introduce a dummy process instance “Code” corresponding to the implementation files: we do not distinguish between the prototype implementation and the final implementation. As events we consider modifications of the corresponding documents, reporting bugs pertaining to them or e-mail communication about them. The developer matching has been constructed manually, using information provided by the software project management plan (prescribed by ESA and available in the Subversion repository). The log obtained with these settings is imported in ProM. Since the management documents do not correspond to a specific development phase of the modified V-model, we filter these process instances and visualize the remaining log using the Dotted Chart visualization. We expect to observe a limited overlap between the series of dots corresponding to adjacent V-model phases.

Figure 4 presents the Dotted Chart visualization obtained for Project IV. For the sake of presentation we have added vertical lines indicating the deadlines stated by the senior management and reflected in the Software Project Management Plan: user requirements phase, software requirements
phase, architecture design phase, detailed design and implementation phase, and two acceptance tests. The size of a dot represents the number of events at the corresponding time. As customary in the V-model design phase, documents (e.g., User Requirements Document) are matched with the corresponding test plans (e.g., Acceptance Test Plan). To visualize this matching we use the same color for dots corresponding to the matching documents.

Figure 4. V-model: sequential development with a limited overlap between the phases and early work on the prototype development are clearly visible.

Figure 4 shows a limited overlap between the adjacent phases. However, the activity peaks, visible as constellations of larger dots, indicate sequential development. Furthermore, closer inspection of Figure 4 reveals that not all test plans were developed according to the V-model: for instance, the Acceptance Test Plan should have been created before the deadline of the User Requirements phase, but was developed throughout the entire project. Relatively late events corresponding to the User Requirements Document and Software Requirements Document represent updates of these documents as the result of change requests.

We have repeated this study for five additional software engineering projects listed in Table 1. In Projects I and VI the user requirements and the software requirements phases overlap, in Projects III and V the software requirements and architectural design phases overlap, and in Projects I and V the architectural and detailed design phases overlap. Project II was exceptional since no significant overlap between the phases has been observed. Moreover, in Project III the overlap between the software requirements and architectural design phases covered almost a half of the time allocated for architectural design. If FRASR+ProM chain would have been available at the time, the senior management could have considered an intervention, e.g., to terminate the work on the Software Requirements Document.

4. Related Work

Capstone projects have been extensively studied in the literature on computer science education [5, 13, 19, 30, 31]. While [30] stresses the importance of a software development process in the classroom and [13] assesses the development process while determining the final grade, the only assessment techniques reported in the literature are evaluation of intermediary artifacts and inherently subjective self- and peer evaluations [5, 13, 19].

Quality assessment of software processes has been a subject of an intensive research and standardization effort (see, e.g., [16]). However, even applying software process assessment models adapted for small companies [29] will require unreasonably high assessment effort from the instructor: evaluation durations of one to three weeks are reported in [29]. As an alternative we propose to augment the existing process assessment by mining software repositories.

Repository-based student software development has been studied in [11]. However, when [11] considered only basic commit patterns in individual assignments as registered in one data source (Subversion), our work addresses a much broader spectrum of software engineering questions, data sources and analysis techniques.

Although mining software repositories is an active research area [6, 12, 14], the research achievements have yet to be applied to assess students’ performance. When compared to the existing repository mining approaches the added value of the FRASR+ProM combination consists in its versatility: multiple data sources can be efficiently combined in multiple ways to produce a multitude of different event logs (FRASR) that in their turn can be be analyzed by applying a wide palet of successful process mining techniques (ProM).

A complementary group of studies has recently applied process mining techniques in the educational setting [15, 20, 27]. These works, however, focussed on learning aspects independent of the learning subject, such as collaborative writing and assessment by means of on-line tests, while our work specifically targets student software development process in capstone projects. Hence, our work had to address specific challenges absent from the aforementioned studies, such as multiplicity of data sources and ways to identify and group events extracted from these data sources.

Both FRASR and ProM have been subject of earlier publications: [21] and [10], respectively. The focus of this paper is, however, on demonstrating how state-of-the-art repository mining can be applied to support the capstone project assessment, rather than on technical details of the solution.

5. Conclusions

Capstone projects are sometimes considered to be crowning achievements of undergraduate software engineering study, amalgamating the knowledge gained in the preceding courses and superseding them in size, complexity and realness. In this paper we have proposed a novel approach to assessment of the development process component of such projects. The approach borrows from the existing research on mining software repositories and business processes. We have shown that the approach proposed allows the instructor to obtain insights in the development process followed without resorting to self-assessments or peer evaluations.
As future work we consider integration of additional software quality assessment tools in preparing intermediate feedback as well as the final evaluation of capstone projects. For instance, while the FRASR+ProM combination can identify separation between the prototype implementation and the final implementation, certain code fragments in the final implementation might have been literally taken from the prototype. To detect this, our approach should be combined with duplication detection techniques [28]. Furthermore, we intend to apply FRASR+ProM to additional projects in the past to create sufficiently large basis for statistical analysis of the way the students carry out the capstone projects. Finally, we plan to conduct a user study, where senior management will be able to give feedback with and without the FRASR+ProM support, and to compare quality of the feedback provided to the students.

References


[27] Vilaythong Southavilay, Kalina Yacef, and Rafael A. Calvo. Process mining to support students’ collaborative writing. In de Baker et al. [7], pages 257–266.


